Title
Adjoint-Based Uncertainty Quantification with MCNP

Author
Seifried, Jeffrey Edwin

Publication Date
2011

Peer reviewed|Thesis/dissertation

Adjoint-Based Uncertainty Quantification with MCNP

by

Jeffrey Edwin Seifried

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Engineering - Nuclear Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Per F Peterson, Chair
Professor Ehud Greenspan
Professor Jasmina L Vujic
Doctor Jeffery F Latkowski
Professor Van P Carey

Fall 2011

Adjoint-Based Uncertainty Quantification with MCNP

by

Jeffrey Edwin Seifried

Abstract

Adjoint-Based Uncertainty Quantification with MCNP

by

Jeffrey Edwin Seifried

Doctor of Philosophy in Engineering - Nuclear Engineering

University of California, Berkeley

Professor Per F Peterson, Chair

This work serves to quantify the instantaneous uncertainties in neutron transport simulations born from nuclear data and statistical counting uncertainties. Perturbation and adjoint theories are used to derive implicit sensitivity expressions. These expressions are transformed into forms that are convenient for construction with MCNP6, creating the ability to perform adjoint-based uncertainty quantification with MCNP6. These new tools are exercised on the depleted-uranium hybrid LIFE blanket, quantifying its sensitivities and uncertainties to important figures of merit. Overall, these uncertainty estimates are small ($< 2\%$). Having quantified the sensitivities and uncertainties, physical understanding of the system is gained and some confidence in the simulation is acquired.

Professor Per F Peterson, Chair Date

Acknowledgements

Lila Chase for assisting in compiling CRSRD, providing MCNP6 source code and compile advice, and squashing a strange bug in MAKXSF. John Wagner for providing the source for his code CRSRD. Massimiliano Fratoni, Antonio Lafuente, Jeff Powers, Tommy Cisneros, Daniel Peters, Alex Hegyi, Tenzing Joshi, Michele Kotiuga, Kevin Kramer, and Ryan Abbott for helpful discussions. Erin Reeves for patience and understanding. Computing support for this work came from the Lawrence Livermore National Laboratory (LLNL) Institutional Computing Grand Challenge program.

"Science is the belief in the ignorance of experts."
—Richard Feynman

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Uncertainty quantification is necessary

The purpose of simulation is to make predictions about a system. If the results of that simulation are excessively uncertain, they can mislead or bring false meaning to conclusions. A responsible simulation effort quantifies the uncertainties–and therefore limitations–of an analysis. Monte Carlo neutron transport calculations depend upon many inputs, such as nuclear data, Monte Carlo methods, isotopic number densities, and system geometries. When those inputs are uncertain, the estimates of results we care about–system thermal power, k_{eff}, reactivity feedback coefficients, flux peaking, radioisotope production, material degradation, and others–are also uncertain. Some estimates of simulation uncertainties are shown in Table 1.1 for a generic fast reactor and Table 1.2 for a generic light-water reactor. These

Simulated Result	**Approximate Uncertainty** [%]
Initial k_{eff}	0.5
Power peaking	3
Power distribution	6
Conversion ratio	6

Table 1.1: Estimates of fast reactor simulation uncertainties [Aliberti *et al.*, 2006].

are just some examples of how neutron transport simulation uncertainties can be excessive, and therefore must be quantified.

Nuclear data describe the manner and probability with which nuclei interact with particles. They are predominately generated with experiments that exhibit some experimental uncertainty and bias and inevitably, experimental estimates of nuclear data do not agree with each other. A simulator has to decide upon a set of nuclear data values before proceed-

Simulated Result	**Approximate Uncertainty** [%]
Power distribution within a fuel pin	8
Power distribution axially within an assembly	4-8
Power distribution radially among assemblies	2-5
Initial k_{eff}	0.3
Reactivity during the lifetime	2-6
^{235}U depletion	5
Ratio of ^{239}Pu to uranium mass	5
Fissile atoms produced	5

Table 1.2: Estimates of light-water reactor simulation uncertainties [Weisbin *et al.*, 1982].

ing to simulate particle transport. Discrepancies between these chosen values and the 'true' values exist and many of these are estimated as dispersions that are tabulated alongside the expected values of nuclear data.

Monte Carlo transport methods sample individual particle histories within a system and aggregate behaviors into expected averages of physical quantity results. The confidence in such results depends upon the amount of information that is available and relevant to that behavior. Mathematically, the coefficient of variation of a result (the standard deviation divided by the expected value) is equal to the number of instances a particle contributes information to the negative one-half power:

$$c_v \equiv \frac{\sigma}{\mu} \approx \frac{1}{\sqrt{N}}, \tag{1.1}$$

where σ is the standard deviation, μ is the expected value, and N is the number of particle instances. Therefore, in order to halve the relative uncertainty, the number of particle contributions to a result must quadruple and by extension four times as many particle histories must be sampled. These Monte Carlo counting statistics are a prime example of aleatory, or random uncertainty, which can be rigorously described with statistics and propagated in models.

Isotopic densities and system geometries are subject to manufacturing tolerances and change over the operation of a system. This work quantifies nuclear data and counting uncertainties within Monte Carlo neutron transport models and these sources of uncertainty were not studied.

1.2 Uncertainty quantification has limitations

The scope of a model is limited to the system it considers. Unforeseen and unconsidered phenomena will always lurk outside of the system boundaries. By corollary, an uncertainty analysis is limited by the uncertainties it considers–the known unknowns. Such an analysis cannot consider all possible uncertainties, so unknown unknowns will always exist. When there is insufficient information regarding an unknown unknown, predictions or conclusions cannot be safely made. There is a danger of devoting the majority of time and resources to studying familiar uncertainties, just to build a false sense of security. More effort should be spent searching for new and diverse uncertainties.

1.3 Literature review

There are several groups that perform excellent work in neutron transport uncertainty quantification. The SCALE neutronics suite out of Oak Ridge National Laboratory [Team, 2011] contains a package TSUNAMI-3D that calculates sensitivities and uncertainties of nuclear systems using adjoint-based methods. For the package's latest update, the breadth of possible analyses expanded from k_{eff} sensitivities and uncertainties to generalized perturbations, for which many types of responses can be considered [Jessee *et al.*, 2009; Rearden *et al.*, 2011] TSUNAMI will be quite powerful when contribution theory is full implemented in the code [Rearden and Williams, 2007; Rearden *et al.*, 2010]. The main advantages this work has over SCALE are that continuous energy simulations are performed with MCNP6 with no approximations or corrections like Dancoff factors, multiple-heterogeneity can be modeled faithfully, transport can be run on distributed parallel systems, and source-driven systems can be modeled.

Adjoint bilinear functional construction looks to be coming to MCNP6 in the near future [Kiedrowski *et al.*, 2011]. It likely will exist first for k_{eff} and critical systems, with eventual extension to general responses and externally sourced systems.

A group out of Seoul National University very thoroughly worked out the theory for propagating explicit sensitivities through a time-dependent Monte Carlo burnup analysis [Park *et al.*, 2011]. The extension of this from explicit to implicit sensitivities would be of enormous benefit.

Great work is also performed at NCSU [Abdel-Khalik *et al.*, 2008], but this is essentially all direct sampling. A Spanish group went beyond the traditional covariance matrix representation of uncertainties to perturb nuclear data parameters [García-Herranz *et al.*, 2008].

1.4 Summary of work

Chapter 2 of this work describes the mathematics behind sensitivity-based uncertainty analyses, outlining its groundings in perturbation theory and adjoint methods. Chapter 3 brings the theory of the previous chapter through the painstaking tedium of building methods to carry out an adjoint-based uncertainty analysis with MCNP6. In Chapter 4, these methods are put into use, quantifying and studying the sensitivities and uncertainties of the depleted-uranium hybrid LIFE blanket. Lastly, Chapter 5 summarizes the conclusions of the work, identifies its successes and shortcomings, and makes suggestions for future work.

1.5 Last thoughts

Those familiar with the artistic works of René Magritte know of his work "The Treachery of Images" in which an ordinary smoking pipe is depicted. Beneath the pipe, which is painted as accurately as possible, is written "This is not a pipe" (translated from French). The reaction of the viewer moves from confusion, to frustration, and finally to resolution with the fact that Magritte was insane. Upon further reflection, the viewer realizes the deeper message. The painting is a mere representation, a depiction a smoking pipe, not the real thing. Perhaps nuclear reactor simulations should be mandated to include the statement "This is not a nuclear reactor."

Chapter 2

Sensitivity and Uncertainty

2.1 Sensitivity-based uncertainty analyses

Sensitivity-based uncertainty analyses allow for the propagation of dispersions in a model's input data to the uncertainties in differential or integral figure of merit neutronic responses. The process requires just two pieces of information: (1) some quantification of the dispersion of the uncertain inputs; and (2) the sensitivities of figures of merit of interest to those input data. Examples of uncertain input data that neutron transport and depletion models are subject to are nuclear cross-sections for a specific reaction of an isotope within a specific neutron energy range, isotopic number densities, and radioactive decay half-lives. The handful of figures of merit of a design can easily depend upon thousands of uncertain inputs.

While the uncertainty of number densities and half-lives can be represented as scalar variances, energy-dependent uncertainties of cross-sections are typically tabulated in covariance matrices which are generated by the practitioners of nuclear data experiments and cross-section evaluators. Sensitivities to these uncertain inputs–energy-dependent vectors for cross-sections and scalars for number densities and half-lives–are problem-specific and represent the bulk of the effort in sensitivity-based uncertainty estimation.

Having quantified the relative covariances of each uncertain input $\mathbf{C}$ and sensitivities of each figure of merit to those inputs S, the uncertainty of that figure of merit U can be computed as the square root of the linear product of sensitivity and relative covariance in quadratic form:

$$U = \sqrt{S\mathbf{C}S^T}. \tag{2.1}$$

In doing so, not only has the confidence in numerical evaluations due to nuclear data uncertainties been estimated, but physical insight to the design has also been provided through sensitivities, and the shortcomings in nuclear data that contribute the most to uncertainties have been identified. An uncertainty analysis is always limited to the variety of uncertainties that are considered–neutronic, thermal fluid, and structural simulation uncertainties can be

quite unrelated.

2.2 Figures of merit and neutronic responses

Figures of merit are metrics which gauge relative performance of designs and are often primary inputs for design or safety analyses. For neutron transport calculations, figures of merit can be derived from (or defined as) system neutronic responses such as the thermal power, tritium production, fissile fuel conversion rates, thermal power peaking, neutron leakage, material damage rates, or reactivity. Physically, these all depend upon the neutron flux; mathematically, they can be represented as responses R which are functionals of response operators $\mathbb{H}_1, \mathbb{H}_2, \ldots, \mathbb{H}_n$ and the neutron flux ψ:

$$R \equiv R[\mathbb{H}_1, \mathbb{H}_2, \ldots, \mathbb{H}_n, \psi]. \tag{2.2}$$

Response operators are typically macroscopic cross-sections with spatial-filter delta functions and the neutron flux is the result of solving the time-independent inhomogeneous neutron transport equation:

$$\mathbb{A}\psi = \mathbb{S}, \tag{2.3}$$

where $\mathbb{A}$ is the transport operator and $\mathbb{S}$ is the external neutron source. In this work, $\mathbb{A}\psi$ is considered the left hand side and $\mathbb{S}$ is considered the right hand side of Equation 2.4:

$$\begin{aligned} &\vec{\Omega} \cdot \nabla\psi(\vec{r}, E, \vec{\Omega}) + \Sigma_t(\vec{r}, E)\ \psi(\vec{r}, E, \vec{\Omega}) \\ -\sum_i \int_0^\infty dE' \int_{4\pi} d\vec{\Omega}'\ &\nu_{x_i}(E')\ \Sigma_{x_i}(\vec{r}, E' \to E, \vec{\Omega}' \to \vec{\Omega})\ \psi(\vec{r}, E', \vec{\Omega}') \\ &= S_{ex}(\vec{r}, E, \vec{\Omega}), \end{aligned} \tag{2.4}$$

where ν_x is the multiplicity of a source reaction: 1 for scattering, $\nu_{fission}$ for fission, 2 for $(n, 2n)$, 3 for $(n, 3n)$ etc. More details on the neutron transport equation can be found in any of the following texts: [Lewins, 1965; Bell and Glasstone, 1970; Lewis and Miller, 1993; Duderstadt and Hamilton, 1976].

All physical quantities in neutron transport models are distributed over some portion of the neutron phase space:

$$\vec{\xi} \equiv (\vec{r}, E, \vec{\Omega}), \tag{2.5}$$

where $\vec{r}$ is the spatial coordinate vector, E is the lab-centered neutron kinetic energy, and $\vec{\Omega}$ is the neutron direction of travel unit vector. In spherical coordinates,

$$\vec{\Omega} \equiv \cos(\varphi)\sin(\theta)\vec{i} + \sin(\varphi)\sin(\theta)\vec{j} + \cos(\theta)\vec{k}, \tag{2.6}$$

where φ and θ are the azimuthal and zenith angle, respectively. A quantity that is invariant with all but $\vec{r}$, either inherently, or after the integration or summation over all other independent dimensions, can be described as a spatial distribution: the thermal power distribution $P_{th}(\vec{r})$ or the scalar neutron flux distribution $\phi(\vec{r})$. A quantity that is invariant with all but E is often referred to as an energy spectrum: the scalar neutron flux spectrum $\phi(E)$ or the sensitivity spectrum $S(E)$. Taking the inner product of a quantity ($\square$ can represent any physical quantity) integrates or sums that quantity over all $\vec{\xi}$:

$$\langle \square \rangle \equiv \int_{\vec{\xi}} d\vec{\xi}\, \square = \int_{\vec{r}} d\vec{r} \int_{E} dE \int_{\vec{\Omega}} d\vec{\Omega}\, \square : \tag{2.7}$$

the total thermal power $P_{th} = \langle P_{th}(\vec{r}) \rangle$ or the total sensitivity $S = \left\langle S(\vec{r}, E, \vec{\Omega}) \right\rangle$. The inner product is a linear functional, so it has the mathematical properties of homogeneity:

$$\langle aX \rangle = a \langle X \rangle , \tag{2.8}$$

and additivity:

$$\langle X + Y \rangle = \langle X \rangle + \langle Y \rangle , \tag{2.9}$$

where X and Y are physical quantities distributed over $\vec{\xi}$ and a is any scalar constant [Renze, 2011].

Any response is somewhat dependent upon (sensitive to) an input parameter, but there is a distinction between explicit and implicit dependence that is important. Explicit dependence (explicit sensitivity) exists only when an input directly contributes to a response by appearing within its response operator $\mathbb{H}$. For example, radiative capture and fission of plutonium are explicit components of calculating the fissile fuel conversion ratio. However, many nuclear reactions affect ψ (indirectly through Equation 2.3). For example, most structural materials cannot produce tritium, but still affect tritium breeding as they tend to depress the neutron economy by parasitically absorbing neutrons that might have otherwise bred tritium. Implicit dependence (implicit sensitivity) captures this effect and others similar to it [Greenspan, 1982]. Mathematically, explicit sensitivity constrains the flux and implicit sensitivity allows it to respond to input perturbations.

2.3 Direct sensitivity and perturbation theory

We define a perturbed quantity that is equal to its nominal value plus a small perturbation:

$$\square' \equiv \square_0 + \delta\square. \tag{2.10}$$

In a system where an input parameter p that the response depends upon explicitly or implicitly is perturbed, the response will also be perturbed. A Taylor expansion can be written for this perturbed response (assuming it is differentiable) with respect to its unperturbed value:

$$R' = R_0 + \frac{1}{1!}\frac{\partial}{\partial p}R\left(p' - p_0\right) + \frac{1}{2!}\frac{\partial^2}{\partial p^2}R\left(p' - p_0\right)^2 + \frac{1}{3!}\frac{\partial^3}{\partial p^3}R\left(p' - p_0\right)^3 + \ldots. \tag{2.11}$$

If either the perturbation in the input parameter is small or the response depends linearly upon it, a first order approximation is sufficient and the expression can be truncated. Doing so brings about a linear approximation for the response perturbation:

$$\delta R = R' - R_0 \cong \frac{\partial R}{\partial p}(p' - p_0) = \frac{\partial R}{\partial p}\delta p. \tag{2.12}$$

Using the chain rule, the partial derivative in Equation 2.12 can be expanded to the partial dependence of the response upon every term its functional (Equation 2.2) contains:

$$\frac{\partial R}{\partial p} = \frac{\partial R}{\partial \mathbb{H}_1}\frac{\partial \mathbb{H}_1}{\partial p} + \frac{\partial R}{\partial \mathbb{H}_2}\frac{\partial \mathbb{H}_2}{\partial p} + \cdots + \frac{\partial R}{\partial \mathbb{H}_n}\frac{\partial \mathbb{H}_n}{\partial p} + \frac{\partial R}{\partial \psi}\frac{\partial \psi}{\partial p}. \tag{2.13}$$

Writing the definition of sensitivity as the linear relative change in a response R with respect to a relative perturbation in an input parameter p:

$$S_{R,p} \equiv \frac{\delta R}{R}\frac{p}{\delta p}, \tag{2.14}$$

it is evident that Equation 2.12 is similar to the sensitivity, modulo a factor of $\frac{p}{R}$. Multiplying by this factor, rearranging, and substituting in Equation 2.13 brings about a first order estimate for the sensitivity of responses of the form of Equation 2.2 to an input parameter:

$$S_{R,p} \cong \frac{\partial R}{\partial p}\frac{p}{R} = \frac{p}{R}\left(\frac{\partial R}{\partial \mathbb{H}_1}\frac{\partial \mathbb{H}_1}{\partial p} + \frac{\partial R}{\partial \mathbb{H}_2}\frac{\partial \mathbb{H}_2}{\partial p} + \cdots + \frac{\partial R}{\partial \mathbb{H}_n}\frac{\partial \mathbb{H}_n}{\partial p} + \frac{\partial R}{\partial \psi}\frac{\partial \psi}{\partial p}\right). \tag{2.15}$$

If the response is a linear functional, consisting of a linear operator (or a linear combination of linear operators) operating on the flux:

$$R = \langle \mathbb{H}\psi \rangle, \tag{2.16}$$

it can be shown that:

$$\frac{\partial R}{\partial \mathbb{H}} = \psi, \tag{2.17}$$

$$\frac{\partial R}{\partial \psi} = \mathbb{H}. \tag{2.18}$$

Substituting these into Equation 2.15, the first-order estimate for sensitivity of linear functionals becomes:

$$S_{R,p} \cong \frac{\Delta \mathbb{H}\psi}{R} + \frac{\mathbb{H}\Delta\psi}{R}, \tag{2.19}$$

where the Δ operator is defined as:

$$\Delta\Box \equiv p\frac{\partial \Box}{\partial p}. \tag{2.20}$$

If the response is a ratio of two linear functionals:

$$R = \frac{\langle \mathbb{H}_1\psi\rangle}{\langle \mathbb{H}_2\psi\rangle}, \tag{2.21}$$

it can be shown that:

$$\frac{\partial R}{\partial \mathbb{H}_1} = \frac{\psi}{\langle \mathbb{H}_2\psi\rangle} = R\frac{\psi}{\langle \mathbb{H}_1\psi\rangle}, \tag{2.22}$$

$$\frac{\partial R}{\partial \mathbb{H}_2} = -\psi\frac{\langle \mathbb{H}_1\psi\rangle}{\langle \mathbb{H}_2\psi\rangle^2} = -R\frac{\psi}{\langle \mathbb{H}_2\psi\rangle}, \tag{2.23}$$

$$\frac{\partial R}{\partial \psi} = \frac{\mathbb{H}_1}{\langle \mathbb{H}_2\psi\rangle} - \langle \mathbb{H}_1\psi\rangle\left(\frac{\mathbb{H}_2}{\langle \mathbb{H}_2\psi\rangle^2}\right) = R\left(\frac{\mathbb{H}_1}{\langle \mathbb{H}_1\psi\rangle} - \frac{\mathbb{H}_2}{\langle \mathbb{H}_2\psi\rangle}\right). \tag{2.24}$$

Substituting these into Equation 2.15, the first-order estimate for sensitivity of ratios of linear functionals becomes:

$$S_{R,p} \cong \left(\frac{\Delta \mathbb{H}_1\psi}{\langle \mathbb{H}_1\psi\rangle} - \frac{\Delta \mathbb{H}_2\psi}{\langle \mathbb{H}_2\psi\rangle}\right) + \left(\frac{\mathbb{H}_1\Delta\psi}{\langle \mathbb{H}_1\psi\rangle} - \frac{\mathbb{H}_2\Delta\psi}{\langle \mathbb{H}_2\psi\rangle}\right). \tag{2.25}$$

In this section, linear perturbation theory was used to derive expressions for direct sensitivities which introduce the Δ operator (Equation 2.20). This operator effectively filters out linear operators that explicitly contain the perturbed input parameter p:

$$\Delta\mathbb{H} \equiv p\frac{\partial \mathbb{H}}{\partial p} = \begin{cases} \mathbb{H}, & \text{if } \mathbb{H} \text{ contains } p \\ 0, & \text{otherwise.} \end{cases} \tag{2.26}$$

This operation occurs in the explicit sensitivity terms which make up the first half of the direct sensitivity expressions. The second half of those expressions are the implicit terms, within which Δ operates upon ψ. ψ, is not a linear operator; it is the solution of Equation

2.3 and can only implicitly depend upon p through its presence in $\mathbb{A}$. Consequently, there is no analytical way to resolve $\Delta\psi$ and more sophisticated approaches are necessary.

2.4 Implicit sensitivity

There exist two prominent strategies in side-stepping the implicit term $\Delta\psi$: direct sampling methods, and adjoint-based methods. In the first, only perturbations are dealt with–input perturbations and the results of perturbed transport calculations–so Δ terms are avoided by producing response sensitivities directly from statistical post-processing. In the second, $\Delta\psi$ terms are replaced with adjoint bilinear functionals and all uncertain inputs are considered in parallel with just one or two adjoint calculations per response.

The stark contrast in computational requirements between the two strategies becomes clear when the number of input parameters and responses are compared. A design can contain hundreds of isotopes, each of which can have just under ten important nuclear reactions that are distributed over just over ten neutron energy decades. Any design contains only a handful of neutronic responses of interest. An analysis using direct sampling methods requires tens of thousands of neutron transport calculations at a minimum while one using adjoint-based methods requires only a handful. For this work, adjoint-based methods are chosen because of their several order of magnitude advantage in computational expense and for other reasons which will soon become clear.

2.4.1 Direct sampling methods

In direct sampling methods, an input is directly perturbed a number times and perturbed responses are extracted from a perturbed transport calculation. A simple linear regression can then produce a linear sensitivity for each response to that input. Technically, only one perturbation is necessary for a linear sensitivity, though more may be desired. Higher-order sensitivity coefficients can also be generated with polynomial regression, requiring many more perturbations than for a linear sensitivity.

The following procedure is repeated for every input of concern:

1. A nuclear data parameter is changed a small amount
2. A transport calculation is performed
3. Responses are extracted from results
4. Response sensitivities are calculated

The extreme regularity and repetition of this brute-force method lends itself to automation through scripting languages, like Perl, BASH, or Python. For example, MCNP6 is distributed

with a Fortran script `mcnp_pstudy` [Brown, 2008] that is marketed for automation of both parameter studies and random sampling of uncertain parameters in a problem. However, the tool is currently limited to perturbing material densities and the random number seed; dispersions in nuclear data are not addressed. Finally, the straight-forward manner in which sensitivities (or deviations) can be generated also makes direct sampling methods convenient for benchmarking of more convoluted methods.

Within an analysis that employs direct sampling methods, there are choices as to how parameters are perturbed. The simplest way is to adjust each parameter by a fractional amount, one at a time, perhaps positively and negatively. This approach is quite limited in the sense that it touches only a small part of the entire parameter space and it does so in a very regular way. It effectively assumes that all parameters are independent. Two much more effective schemes are Latin Hypercube sampling and orthogonal sampling, for which it is assured that all regions of phase space are sampled appropriately [Iman and Conover, 1982].

2.4.2 Adjoint theory

Historical accounts of the invention of variational methods inevitably mention the brachistochrone problem, for which the curve of fastest descent under the acceleration of gravity is sought. In solving this problem, a new field of mathematics, 'Calculus of Variations,' was born, offering tools to streamline the pursuit and study of extrema and stationarities. Variational methods are used in the fields of mechanics, geometry, economics, control theory, theoretical physics, and others.

In neutronics, variational methods can derive an adjoint neutron transport equation, whose solution, the adjoint neutron flux, is the dual of the forward neutron flux. Solving for one requires no knowledge of the other; both solutions depend upon the system geometry and materials, but the adjoint neutron flux is additionally solved with respect to a defined response. For each response the resultant adjoint distribution (or importance) is equivalent to the probability of neutron sources or sinks in regions of neutron phase space in increasing or decreasing that response, respectively. In other words, the importance quantifies the potential effectiveness of neutrons in causing the response, depending upon where they originate within phase space.

For example, consider the neutron flux detector in Figure 2.1 with a neutronic response defined as the rate of (n, γ) reactions, surrounded by two optically thin physical regions. If the response importance of region A is twice that of region B, neutrons that are born in region A are twice as likely to contribute to the response. Contrariwise, neutrons lost in region B are half as likely to diminish the (n, γ) reading on the detector as those lost in region A. Synthesizing, neutron source and sink perturbations in region A are twice as significant in perturbing the detector reading as neutron source and sink perturbations in region B. The adjoint distribution enables one to quantify the implicit effect that perturbations have

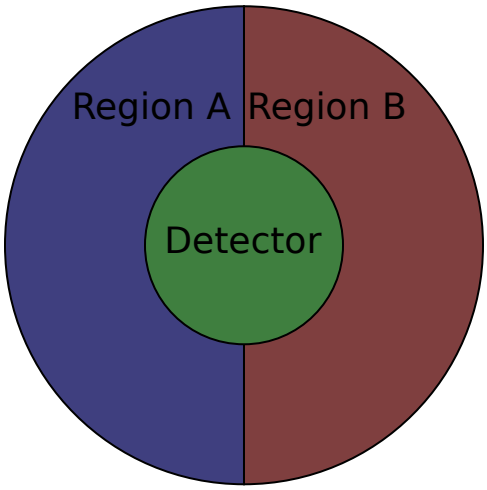

Figure 2.1: An example detector response surrounded by two regions.

on a response, by way of their influence upon the flux [Selengut, 1959; Lewins, 1965; Stacey, 1974; Greenspan, 1976a; Coveyou *et al.*, 1967].

In the next section, the variational method of Lagrange multipliers is used to derive the adjoint neutron transport equation for a neutronic response. The solution of this equation produces the adjoint distribution, or importance. The powerful ability of the adjoint to simplify expressions containing $\Delta\psi$ terms is then demonstrated using this variational method and a differential method [Williams, 1982].

2.4.2.1 Derivation with variational methods

A Lagrange functional L can be written for a response functional R, using the forward time-independent inhomogeneous neutron transport equation (Equation 2.3) as a constraint and Lagrange multiplier λ:

$$L \equiv R - \langle \lambda(\mathbb{A}\psi - \mathbb{S}) \rangle \,. \tag{2.27}$$

The functional is nominally equal to the response due to nominal satisfaction of the constraint and can be considered explicitly dependent only upon the Lagrange multiplier, the neutron flux, and the transport operator which will be represented by an arbitrary input parameter p:

$$L = L[\lambda, \psi, p] = R. \tag{2.28}$$

Similarly, upon perturbation of the input parameter, a perturbed functional can be written:

$$L' = L[\lambda', \psi', p'] = R'. \tag{2.29}$$

A Taylor expansion (terminating at first order) can be written for the perturbed functional

with respect to the unperturbed functional:

$$L' \cong L + \left\langle \frac{\partial L}{\partial \lambda}\delta\lambda + \frac{\partial L}{\partial \psi}\delta\psi + \frac{\partial L}{\partial p}\delta p \right\rangle. \tag{2.30}$$

Combining the three previous equations, the first variation in the response functional is found:

$$\delta R \equiv R' - R_0 \cong \left\langle \frac{\partial L}{\partial p}\delta p \right\rangle\Big|_{\lambda,\psi} + \left\langle \frac{\partial L}{\partial \lambda}\delta\lambda \right\rangle\Big|_{p,\psi} + \left\langle \frac{\partial L}{\partial \psi}\delta\psi \right\rangle\Big|_{p,\lambda}. \tag{2.31}$$

If the first variation of the response functional is stationary about the Lagrange multiplier and flux (i.e. if the last two terms in Equation 2.31 are zero), it can simplify to only those terms that explicitly depend upon the perturbation. Stationarity about the Lagrange multiplier requires:

$$\left\langle \frac{\partial L}{\partial \lambda}\delta\lambda \right\rangle\Big|_{p,\psi} = \langle (\mathbb{A}\psi - \mathbb{S})\delta\lambda \rangle = 0, \tag{2.32}$$

or for an arbitrary $\delta\lambda$:

$$\mathbb{A}\psi - \mathbb{S} = 0, \tag{2.33}$$

which is satisfied by Equation 2.3. Stationarity about the neutron flux requires:

$$\left\langle \frac{\partial L}{\partial \psi}\delta\psi \right\rangle\Big|_{p,\lambda} = \left\langle \frac{\partial R}{\partial \psi}\delta\psi \right\rangle - \langle \lambda \mathbb{A}\delta\psi \rangle = 0, \tag{2.34}$$

or applying the commutativity relation of an adjoint operator $\langle x, \mathbb{B}y \rangle = \langle y, \mathbb{B}^\dagger x \rangle$ [Weisstein, 2011a]:

$$\left\langle \left(\frac{\partial R}{\partial \psi} - \mathbb{A}^\dagger \lambda \right) \delta\psi \right\rangle = 0, \tag{2.35}$$

or for an arbitrary $\delta\psi$:

$$\mathbb{A}^\dagger \lambda = \frac{\partial R}{\partial \psi}. \tag{2.36}$$

This equation has the same form as the forward time-independent inhomogeneous neutron transport equation (Equation 2.3), with the adjoint of the transport operator $\mathbb{A}^\dagger$ in place of the forward transport operator, the Lagrange multiplier (hereafter referred to as the adjoint flux $\psi^\dagger$) in place of the neutron flux, and a response partial derivative (hereafter referred to as the adjoint source $\mathbb{S}^\dagger$) in place of the forward external neutron source. Stationarity of the response functional about the neutron flux requires that the adjoint flux satisfy the adjoint time-independent inhomogeneous neutron transport equation:

$$\mathbb{A}^\dagger \psi^\dagger = \frac{\partial R}{\partial \psi} \equiv \mathbb{S}^\dagger. \tag{2.37}$$

In this work, $\mathbb{A}^\dagger\psi^\dagger$ is considered the left hand side and $\mathbb{S}^\dagger$ is considered the right hand side of Equation 2.38:

$$\begin{aligned} -\vec{\Omega}\cdot\nabla\psi^\dagger(\vec{r},E,\vec{\Omega}) + \Sigma_t(\vec{r},E)\ \psi^\dagger(\vec{r},E,\vec{\Omega}) \\ -\sum_i \nu_{x_i}(E) \int_{4\pi} d\vec{\Omega}' \int_0^\infty dE'\ \Sigma_{x_i}(\vec{r},E\to E',\vec{\Omega}\to\vec{\Omega}')\ \psi^\dagger(\vec{r},E',\vec{\Omega}') \\ = S_{ex}(\vec{r},E,\vec{\Omega}). \end{aligned} \tag{2.38}$$

All terms are consistent with their representation in Equation 2.4. More details on the transport equation can be found in any of the following texts: [Greenspan, 1976b; Lewis and Miller, 1993].

With a neutron flux that satisfies Equation 2.3 and an adjoint flux that satisfies Equation 2.37, the Lagrange functional is stationary about all but p (the first term in Equation 2.31):

$$\delta R \cong \left\langle \frac{\partial L}{\partial p}\delta p \right\rangle\Big|_{\lambda,\psi} = \left\langle \left(\frac{\partial R}{\partial p}\Big|_{\psi} - \psi^\dagger\frac{\partial \mathbb{A}}{\partial p}\psi + \psi^\dagger\frac{\partial \mathbb{S}}{\partial p}\right)\delta p \right\rangle, \tag{2.39}$$

or using Equation 2.13:

$$\delta R \cong \left\langle \frac{\partial R}{\partial p}\delta p \right\rangle - \left\langle \left(\frac{\partial R}{\partial \psi}\frac{\partial \psi}{\partial p} - \left(-\psi^\dagger\frac{\partial \mathbb{A}}{\partial p}\psi + \psi^\dagger\frac{\partial \mathbb{S}}{\partial p}\right)\right)\delta p \right\rangle. \tag{2.40}$$

When this equation is compared to Equation 2.12, it is clear that the second term is zero, or for arbitrary δp:

$$\frac{\partial R}{\partial \psi}\frac{\partial \psi}{\partial p} = -\psi^\dagger\frac{\partial \mathbb{A}}{\partial p}\psi + \psi^\dagger\frac{\partial \mathbb{S}}{\partial p}. \tag{2.41}$$

This relation allows for the elimination of $\Delta\psi$ by substituting in more manageable terms in which Δ is operating only upon linear operators. In the next section, this relation is derived using differential methods.

2.4.2.2 Derivation with differential methods

Writing the forward time-independent inhomogeneous neutron transport equation (Equation 2.3) for a perturbed system:

$$(\mathbb{A}+\delta\mathbb{A})(\psi+\delta\psi) = \mathbb{S}+\delta\mathbb{S}, \tag{2.42}$$

the unperturbed terms can be eliminated with Equation 2.3 and the second-order perturbation terms can be neglected:

$$\mathbb{A}\delta\psi = -\delta\mathbb{A}\psi + \delta\mathbb{S}. \tag{2.43}$$

Subtracting the inner product of the flux perturbation $\delta\psi$ with Equation 2.37 from the inner product of the adjoint flux $\psi^\dagger$ with Equation 2.43:

$$\left\langle \psi^\dagger, \mathbb{A}\delta\psi \right\rangle - \left\langle \delta\psi, \mathbb{A}^\dagger\psi^\dagger \right\rangle = -\left\langle \psi^\dagger, \delta\mathbb{A}\psi \right\rangle + \left\langle \psi^\dagger\delta\mathbb{S} \right\rangle - \left\langle \frac{\partial R}{\partial\psi}\delta\psi \right\rangle. \tag{2.44}$$

Applying the commutativity relation of an adjoint operator $\langle x, \mathbb{B}y\rangle = \left\langle y, \mathbb{B}^\dagger x\right\rangle$, the first two terms can be canceled:

$$\left\langle \frac{\partial R}{\partial\psi}\delta\psi \right\rangle = -\left\langle \psi^\dagger, \delta\mathbb{A}\psi \right\rangle + \left\langle \psi^\dagger\delta\mathbb{S} \right\rangle. \tag{2.45}$$

When perturbations are truncated to first order like in Equation 2.12, Equation 2.41 emerges, showing that the variational and differential derivations are equivalent.

2.4.2.3 Adjoint-based sensitivity

Taking advantage of the tranformative properties of the adjoint flux exemplified in Equation 2.41, a new first-order adjoint-based sensitivity for responses with the form of Equation 2.2 is found to replace the direct version of Equation 2.15:

$$S_{R,p} \cong \frac{\frac{\partial R}{\partial \mathbb{H}_1}\Delta\mathbb{H}_1}{R} + \frac{\frac{\partial R}{\partial \mathbb{H}_2}\Delta\mathbb{H}_2}{R} + \cdots + \frac{\frac{\partial R}{\partial \mathbb{H}_n}\Delta\mathbb{H}_n}{R} - \frac{\psi^\dagger\Delta\mathbb{A}\psi}{R} + \frac{\psi^\dagger\Delta\mathbb{S}}{R}. \tag{2.46}$$

For linear functional responses with the form of Equation 2.16, the first-order estimate for sensitivity becomes:

$$S_{R,p} \cong \left(\frac{\Delta\mathbb{H}\psi}{\langle\mathbb{H}\psi\rangle}\right) - \frac{\psi^\dagger\Delta\mathbb{A}\psi}{R} + \frac{\psi^\dagger\Delta\mathbb{S}}{R}, \tag{2.47}$$

with adjoint source:

$$\mathbb{S}^\dagger = \mathbb{H} = \left(\frac{R}{\langle\mathbb{H}\psi\rangle}\right)\mathbb{H}. \tag{2.48}$$

For ratios of linear functional responses with the form of Equation 2.21, the first-order estimate for sensitivity becomes:

$$S_{R,p} \cong \left(\frac{\Delta\mathbb{H}_1\psi}{\langle\mathbb{H}_1\psi\rangle} - \frac{\Delta\mathbb{H}_2\psi}{\langle\mathbb{H}_2\psi\rangle}\right) - \frac{\psi^\dagger\Delta\mathbb{A}\psi}{R} + \frac{\psi^\dagger\Delta\mathbb{S}}{R}, \tag{2.49}$$

with adjoint source:

$$\mathbb{S}^\dagger = R\left(\frac{\mathbb{H}_1}{\langle\mathbb{H}_1\psi\rangle} - \frac{\mathbb{H}_2}{\langle\mathbb{H}_2\psi\rangle}\right) = \left(\frac{R}{\langle\mathbb{H}_1\psi\rangle}\right)\mathbb{H}_1 - \left(\frac{R}{\langle\mathbb{H}_2\psi\rangle}\right)\mathbb{H}_2. \tag{2.50}$$

With the adjoint flux and an $\Delta\mathbb{A}$ that can be expressed analytically, the implicit effects of any and all input perturbations can be quantified with respect to a nominal state without ever having to calculate a single perturbed state or $\Delta\psi$.

2.5 Material composition optimization

The utility of sensitivities is not limited to propagating uncertainties; they can also be a tool for optimization. If the input parameter p is the atomic density of an isotope, the total sensitivity of a response R to that input parameter $S_{R,p}$ quantifies the linear effect of changes in that isotope's density upon the response. If this sensitivity is positive, increments or decrements in the density will increase or decrease the response, respectively. Negative sensitivities indicate the opposite and responses are essentially independent of (insensitive to) the density if this sensitivity is small or zero. If extremal responses are desired, isotopic density adjustments can be performed iteratively with sensitivity estimations until all sensitivities are small. This can be especially effective when isotopic densities and sensitivities are distributed spatially. Physical insight can also be garnered from the sensitivity distribution [Greenspan, 1982; Cacuci, 2003].

Because of properties of the Δ operator described in Equation 2.26, when p is an isotopic atomic density, Δ hits all source and sink cross-sections for that isotope. Additionally, due to their relative senses, sensitivities to macroscopic and microscopic cross-sections are identical. Consequently, the sensitivity of a response to an isotopic atomic density is equivalent to the sum of the sensitivities of that response to the cross-sections (macroscopic or microscopic) for that isotope of every reaction type:

$$S_{R,z} = \sum_i S_{R,zx_i}, \tag{2.51}$$

where z and x_i denote the isotope index and reaction type index, respectively. Often isotopic density optimization is not possible because the isotopes of interest are contained within a single element or chemical compound. In these cases, the individual isotopic sensitivities must be combined according to their relative atomic abundances within the material:

$$S_{R,m} = \frac{\sum_i a_i S_{R,z_i}}{\sum_i a_i}, \tag{2.52}$$

where m and a_i denote the material index and the atomic abundances, respectively.

Chapter 3

Sensitivity Estimation with MCNP6

In Chapter 2, adjoint-based sensitivities were derived for two types of responses: (1) linear functionals of the flux; and (2) ratios of linear functionals of the flux. Ultimately, the terms necessary to construct these sensitivities must be extracted from neutron transport simulations. This chapter describes that process for the three-dimensional, continuous energy, true heterogeneous, Monte Carlo neutron transport code MCNP6 [X-5 Monte Carlo Team, 2005]. Upon inspection, three types of terms emerge within these sensitivities (Equations 2.47 and 2.49): (1) response operator linear functionals $\Delta\mathbb{H}\psi$ (or equivalently $\mathbb{H}\psi$); (2) transport operator bilinear functionals $\psi^\dagger\Delta\mathbb{A}\psi$; and (3) source perturbation linear functionals $\psi^\dagger\Delta\mathbb{S}$. In the absence of source perturbations in this work, source perturbation linear functionals are zero and consequently are ignored. The remaining two terms are addressed in turn.

Explicit sensitivity (for which implicit terms are neglected) requires just the construction of response operator linear functionals. Implicit sensitivity extraction is quite a bit more involved since it additionally requires construction of the transport operator bilinear functionals, which are difficult to extract. Each of the necessary steps is described in the section dedicated to construction of the transport operator bilinear functional. First, it is expanded into more manageable terms, introducing the angular distribution. After the angular distribution is studied in depth and discretized, angle smearing is solved for curve-linear spherical coordinate systems. Neutron sinks and sources are then discretized using tally tags and all the pieces are assembled to calculate implicit sensitivities. Source code is provided for algorithms when it is appropriate.

Afterwards, a patch is described, which enables MCNP6 to tally source distributions for particles 'born' from elastic scatters. Then, proper construction of adjoint sources and subsequent tallying of adjoint distributions is shown. Proper extraction of figure of merit responses and propagation of counting uncertainties onto nuclear data uncertainties are then discussed. Finally, generation of multi-group cross-section libraries for adjoint transport calculations and nuclear data covariances are described.

3.1 Constructing the linear functional

When $\mathbb{H}$ is a macroscopic cross-section, $\Delta\mathbb{H}\psi$ is a reaction rate distribution, shown here as distributed over $(\vec{r}, E)$ in neutron phase space:

$$\begin{aligned}\Delta\mathbb{H}\psi &\equiv \int_{\vec{\Omega}} d\vec{\Omega}\; p\frac{\partial\mathbb{H}}{\partial p}(\vec{r}, E)\; \psi(\vec{r}, E, \vec{\Omega}) \\ &= \int_{\vec{\Omega}} d\vec{\Omega}\; \Sigma_{z,x}(\vec{r}, E)\; \psi(\vec{r}, E, \vec{\Omega}) \\ &= \Sigma_{z,x}(\vec{r}, E) \int_{\vec{\Omega}} d\vec{\Omega}\; \psi(\vec{r}, E, \vec{\Omega}) \\ &= \Sigma_{z,x}(\vec{r}, E)\; \phi(\vec{r}, E) \\ &= R_{z,x}(\vec{r}, E),\end{aligned} \tag{3.1}$$

where z and x are the indices for the isotope and reaction type of input parameter p, respectively. In a discrete sense, this can be extracted directly from MCNP6 with a standard F^{M4} cell flux multiplier tally for multiple cells, binned over neutron energy [X-5 Monte Carlo Team, 2005, p. 2-105]:

$$\Delta\mathbb{H}\psi = F^{M4}_{c,z,x}(E), \tag{3.2}$$

where c is the cell index of input parameter p. The values of the linear functional within each cell and energy bin (per unit bin width) are effective averages for those discrete regions in phase space, which converge to the true continuous values as the bin size is shrunk to arbitrarily small size. The collapse over $\vec{\xi}$ is not affected by binning scheme and subsequently, there are no discretization errors in the evaluation of the total linear functional $\langle\Delta\mathbb{H}\psi\rangle$ with F^{M4} tallies.

The algorithm for extracting explicit sensitivities for a given response iterates over all F^{M4} cell flux multiplier tallies within a forward MCNP6 calculation, finding those that match with constituents of the response and adjusting for the required multiplicity and sign. Depending on the purpose of the sensitivity, it can be accumulated over cells, isotopes, reactions, or combinations thereof, for possible indices of (c, z, x), (c, z), (z, x), and (z). The results are then divided by the total linear functional and multiplied by the response, consistent with the explicit sensitivity expressions for both types of responses (see Equations 2.47 and 2.49). Specific details of the algorithm can be found in the `GetResponseExplicitSensitivitys` method of the `McnpOutputFile` class within `ParseMcnp.py`:

Algorithm 3.1: GetResponseExplicitSensitivitys

```
1    def GetResponseExplicitSensitivitys(self, response, sumOverCells, sumOverZas,
         sumOverReactions):
```

```
2           ###
3           assert(self.GetIsForward());
4           ###
5           # Total dividend and divisor
6           ###
7           totals = (self.GetTotal(response.GetDividend()), self.GetTotal(response.
                GetDivisor()));
8           ###
9           # Perturbation
10          ###
11          perturbation = response.GetPerturbation();
12          isPerturbation = bool(perturbation);
13          if isPerturbation:
14              perturbCellNumber, perturbZa, perturbReactionNumber = perturbation[0];
15          ###
16          # Iterate over reaction rate tallys
17          ###
18          key2ExplicitSensitivity = {};
19          for cellNumber, multiplierBins in self.GetTallyIndices('fm4').items():
20              ###
21              # Kick out immaterial cells
22              # Only cells with materials can contribute to explicit sensitivity
23              ###
24              if not self.FindCell(cellNumber).GetMaterialNumber():
25                  continue;
26              ###
27              # Iterate over multiplier bins
28              ###
29              for materialNumber, reactionNumber in multiplierBins:
30                  try:
31                      ###
32                      # Extract single-za from material number
33                      ###
34                      za = self.GetMaterialNumber2SingleZa()[materialNumber];
35                  except KeyError:
36                      ###
37                      # Kick out non-single-za material numbers
38                      ###
39                      continue;
40                  ###
41                  # Kick out if indices don't match perturbation indices
42                  ###
43                  if isPerturbation:
44                      # c
45                      if perturbCellNumber not in (None, cellNumber):
46                          continue;
47                      # z
48                      if perturbZa not in (None, za):
49                          continue;
50                      # x
51                      if perturbReactionNumber not in (None, reactionNumber):
52                          continue;
53                  ###
54                  # Extract multiplicitys of (cell number, material number, reaction
                        number) within response
55                  ###
56                  multiplicitys = response.GetMultiplicities(cellNumber, materialNumber,
                         reactionNumber);
57                  ###
58                  # Iterate over dividend, divisor
```

```
59                  ###
60                  for index in range(2):
61                      multiplicity = multiplicitys[index];
62                      ###
63                      # Kick out (cell number, material number, reaction number) with
                            null multiplicity
64                      ###
65                      if not multiplicity:
66                          continue;
67                      ###
68                      # Explicit sensitivity distribution
69                      ###
70                      explicitSensitivity = SafeDivide(self.
                            GetCellReactionRateEnergyDistribution(cellNumber,
                            materialNumber, reactionNumber), totals[index] / multiplicity)
                            ;
71                      ###
72                      # Square away summation indices
73                      ###
74                      key = [];
75                      ###
76                      if not sumOverCells:
77                          ###
78                          # Ascend to root cell
79                          ###
80                          for cell in self.FindRootCells(cellNumber):
81                              cellNumber = cell.GetNumber();
82                              break;
83                          key.append(cellNumber);
84                      ###
85                      if not sumOverZas:
86                          key.append(za);
87                      ###
88                      if not sumOverReactions:
89                          key.append(reactionNumber);
90                      ###
91                      key = tuple(key);
92                      ###
93                      try:
94                          key2ExplicitSensitivity[key] += explicitSensitivity;
95                      except KeyError:
96                          key2ExplicitSensitivity[key] = explicitSensitivity;
97              ###
98              # Check explicit sensitivity discrepancies
99              ###
100             discrepancy = sum(1. - 2. * index for index in range(2) if totals[index]) -
                    sum(ed.GetTotalElement() for ed in key2ExplicitSensitivity.values() if ed)
101             if abs(discrepancy) > 1e-3:
102                 Warning('{:+.2%} of explicit sensitivity is unaccounted for in response
                        `{}\' with single-isotope reaction rates: {}!'.format(discrepancy,
                        response, ', '.join(str(key) for key in sorted(key2ExplicitSensitivity
                        ))));
103             ###
104             return key2ExplicitSensitivity;
```

3.2 Constructing the bilinear functional

MCNP6 is an extremely versatile particle transport simulation code, capable of modeling the interaction of nearly any type of radiation with nearly everything, with a large range of tally outputs. It is not, however designed to construct adjoint bilinear functionals. Because of this, the bulk of response calculations, sensitivity estimation, and uncertainty propagation must be done externally to MCNP6 in a suite of Python scripts entitled PSABUQM (Python Suite for Adjoint-Based Uncertainty Quantification with MCNP6).

3.2.1 Angular segregation of the adjoint distribution

The bilinear functional $\psi^\dagger \Delta\mathbb{A}\psi$ is by no means separable in $\vec{r}$, E, and $\vec{\Omega}$. However, when its distribution over $(\vec{r}, E)$ is ultimately sought-after, some segregation of angular dependence from the rest is beneficial. First, angular segregation of the adjoint angular flux is performed:

$$\begin{aligned}
\psi^\dagger \Delta\mathbb{A}\psi &\equiv \int_{\vec{\Omega}} d\vec{\Omega}\ \psi^\dagger(\vec{r}, E, \vec{\Omega})\ \Delta\mathbb{A}\psi(\vec{r}, E, \vec{\Omega}) \\
&= \frac{\int_{\vec{\Omega}} d\vec{\Omega}\ \psi^\dagger(\vec{r}, E, \vec{\Omega})}{\int_{\vec{\Omega}} d\vec{\Omega}\ \psi^\dagger(\vec{r}, E, \vec{\Omega})} \int_{\vec{\Omega}} d\vec{\Omega}\ \psi^\dagger(\vec{r}, E, \vec{\Omega})\ \Delta\mathbb{A}\psi(\vec{r}, E, \vec{\Omega}) \\
&= \frac{\phi^\dagger(\vec{r}, E)}{\phi^\dagger(\vec{r}, E)} \int_{\vec{\Omega}} d\vec{\Omega}\ \psi^\dagger(\vec{r}, E, \vec{\Omega})\ \Delta\mathbb{A}\psi(\vec{r}, E, \vec{\Omega}) \\
&= \phi^\dagger(\vec{r}, E) \int_{\vec{\Omega}} d\vec{\Omega}\ \frac{\psi^\dagger(\vec{r}, E, \vec{\Omega})}{\phi^\dagger(\vec{r}, E)}\ \Delta\mathbb{A}\psi(\vec{r}, E, \vec{\Omega}) \\
&= \phi^\dagger(\vec{r}, E) \int_{\vec{\Omega}} d\vec{\Omega}\ \alpha^\dagger(\vec{r}, E, \vec{\Omega})\ \Delta\mathbb{A}\psi(\vec{r}, E, \vec{\Omega}),
\end{aligned} \tag{3.3}$$

where α is the normalized angular distribution for every spatial and energy value, defined as:

$$\alpha(\vec{r}, E, \vec{\Omega}) \equiv \frac{\psi(\vec{r}, E, \vec{\Omega})}{\phi(\vec{r}, E)}. \tag{3.4}$$

3.2.2 Discretizing the angular distribution

Before α can be discretized, some analysis must be done to derive relations between angular and scalar flux and current. All quantities are then shown to be constructible from F^1 surface current tallies, which is the only current tool for extracting angular-energy correlation of particles in MCNP6 [X-5 Monte Carlo Team, 2005, p. 2-80].

The angular neutron current $\vec{J}(\vec{r}, E, \vec{\Omega})$ can be written in terms of the angular neutron flux $\psi(\vec{r}, E, \vec{\Omega})$:

$$\vec{J}(\vec{r}, E, \vec{\Omega}) \equiv \vec{\Omega}\psi(\vec{r}, E, \vec{\Omega}), \tag{3.5}$$

where both $\vec{J}(\vec{r}, E, \vec{\Omega})$ and $\psi(\vec{r}, E, \vec{\Omega})$ have units of $\left[\frac{\text{neutron}}{\text{cm}^2\cdot\text{source}\cdot\text{MeV}\cdot\text{ster}}\right]$. If $\vec{J}(\vec{r}, E, \vec{\Omega})$ is invariant with the azimuthal angle φ and directionally a function of only the zenith angle cosine $\mu \equiv \cos(\theta)$, then:

$$\vec{J}(\vec{r}, E, \vec{\Omega}) = \frac{1}{2\pi}\int_0^{2\pi} d\varphi\ \vec{J}(\vec{r}, E, \vec{\Omega}) = \frac{\vec{J}(\vec{r}, E, \theta)}{2\pi} = \frac{\vec{J}(\vec{r}, E, \mu)}{2\pi}; \tag{3.6}$$

equivalently the same thing can be said for $\psi(\vec{r}, E, \vec{\Omega})$:

$$\psi(\vec{r}, E, \vec{\Omega}) = \frac{1}{2\pi}\int_0^{2\pi} d\varphi\ \psi(\vec{r}, E, \vec{\Omega}) = \frac{\psi(\vec{r}, E, \theta)}{2\pi} = \frac{\psi(\vec{r}, E, \mu)}{2\pi}, \tag{3.7}$$

where both $\vec{J}(\vec{r}, E, \mu)$ and $\psi(\vec{r}, E, \mu)$ have units of $\left[\frac{\text{neutron}}{\text{cm}^2\cdot\text{source}\cdot\text{MeV}\cdot\text{abin}}\right]$. Putting the three previous equations together:

$$\begin{aligned}\vec{J}(\vec{r}, E, \mu) &= \int_0^{2\pi} d\varphi\ \vec{J}(\vec{r}, E, \vec{\Omega}) = \int_0^{2\pi} d\varphi\ \vec{\Omega}\psi(\vec{r}, E, \vec{\Omega}) \\ &= \int_0^{2\pi} d\varphi\ \vec{\Omega}\ \frac{\psi(\vec{r}, E, \mu)}{2\pi} = 2\pi\mu\vec{k}\frac{\psi(\vec{r}, E, \mu)}{2\pi},\end{aligned} \tag{3.8}$$

where $\vec{k}$ is the zenith directional unit vector, or:

$$\psi(\vec{r}, E, \mu) = \frac{\left|\vec{J}(\vec{r}, E, \mu)\right|}{\mu}. \tag{3.9}$$

In MCNP6, F^1 surface current tallies count the number of neutrons crossing a surface, binned by energy and angle, with units of $\left[\frac{\text{neutron}}{\text{source}\cdot\text{ebin}\cdot\text{abin}}\right]$ [X-5 Monte Carlo Team, 2005, p.

2-81]:

$$\begin{aligned} F_s^1(E,\mu) &\equiv \int_{\Delta A} dA \int_{\Delta E} dE \int_{\Delta\mu} d\mu \int_0^{2\pi} d\varphi \; \vec{J}(\vec{r},E,\vec{\Omega})\cdot\vec{k} \\ &= \int_{\Delta A} dA \int_{\Delta E} dE \int_{\Delta\mu} d\mu \; \vec{J}(\vec{r},E,\mu)\cdot\vec{k}. \end{aligned} \tag{3.10}$$

The average angular neutron current within the discrete region of phase space defined by surface area ΔA on surface s, energy bin width ΔE about E, and directional angle bin width $\Delta\mu$ about μ per unit of phase space follows:

$$\left|\vec{J}(\vec{r},E,\mu)\right| \cong \frac{F^1(\vec{r},E,\mu)}{\Delta A\;\Delta E\;\Delta\mu}. \tag{3.11}$$

Using Equation 3.9, this can be extended to angular flux:

$$\psi(\vec{r},E,\mu) \cong \frac{F^1(\vec{r},E,\mu)}{\mu\;\Delta A\;\Delta E\;\Delta\mu}. \tag{3.12}$$

Rewriting the scalar flux in terms of the angular flux:

$$\begin{aligned} \phi(\vec{r},E) &\equiv \int_{4\pi} d\vec{\Omega}\;\psi(\vec{r},E,\vec{\Omega}) = \int_0^{\pi} d\theta\;\sin(\theta)\;\frac{\psi(\vec{r},E,\theta)}{2\pi}\int_0^{2\pi} d\varphi \\ &= \int_0^{\pi} d\theta\sin(\theta)\;\psi(\vec{r},E,\theta) = \int_{-1}^{+1} d\mu\;\psi(\vec{r},E,\mu), \end{aligned} \tag{3.13}$$

or in discrete form:

$$\phi(\vec{r},E) = \sum_i \Delta\mu_i\;\psi(\vec{r},E,\mu_i). \tag{3.14}$$

From Equations 3.12, 3.13, and 3.7, α (Equation 3.4) can be constructed from F^1 surface

current tallies:

$$\begin{aligned}\alpha(\vec{r},E,\vec{\Omega}) \equiv \frac{\psi(\vec{r},E,\vec{\Omega})}{\phi(\vec{r},E)} &= \frac{\frac{1}{2\pi}\psi(\vec{r},E,\mu)}{\int_{-1}^{+1} d\mu\ \psi(\vec{r},E,\mu)} \\ &= \frac{\frac{1}{2\pi}\frac{F^1(\vec{r},E,\mu)}{\mu\ \Delta A\ \Delta E\ \Delta\mu}}{\int_{-1}^{+1} d\mu\ \frac{F^1(\vec{r},E,\mu)}{\mu\ \Delta A\ \Delta E\ \Delta\mu}} = \frac{\frac{F^1(\vec{r},E,\mu)}{\mu\ 2\pi\ \Delta\mu}}{\int_{-1}^{+1} d\mu\ \frac{F^1(\vec{r},E,\mu)}{\mu\ \Delta\mu}},\end{aligned} \tag{3.15}$$

or in a discrete form:

$$\alpha_s(E,\vec{\Omega}_i) = \frac{\frac{F_s^1(E,\mu_i)}{\bar{\mu}_i\ 2\pi\ \Delta\mu_i}}{\sum_j \Delta\mu_j\ \frac{F_s^1(E,\mu_j)}{\bar{\mu}_j\ \Delta\mu_j}} = \frac{\frac{F_s^1(E,\mu_i)}{\bar{\mu}_i\ 2\pi\Delta\mu_i}}{\sum_j \frac{F_s^1(E,\mu_j)}{\bar{\mu}_j}}, \tag{3.16}$$

where s is the surface index. With Equation 3.16, α can be constructed over any surface. However, bilinear functionals are calculated over cells and not surfaces, so the angular distributions for cells must be related to those of the surfaces that bound the cell.

3.2.3 "Angle smearing": mapping surface directionality to cells

Cells in MCNP6 are defined according to boundary surfaces. Consequently, cell angular distributions can receive contributions from multiple surfaces–particles are accumulated over the energy and angular bins of each F^1 surface current tally. A complication arises when the global coordinate system over which physical quantities tend to vary is globally curve-linear. Figure 3.1 shows how directionality changes with position within such a coordinate system. The global zenith angle θ_g (with origin at the sphere centers) changes continuously for a

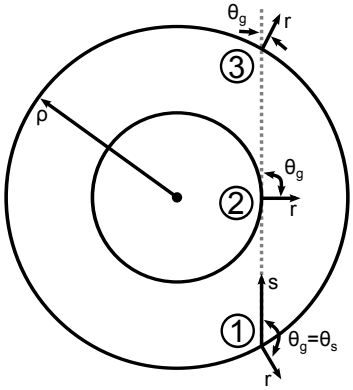

Figure 3.1: Spatial dependence of direction within a curve-linear coordinate system.

particle that traverses the dotted line upwards: at ① $\cos(\theta_g)$ is negative, at ② it is zero, and at ③ it is positive. The particle crosses the surface at ① with a single direction, but has a continuous range of global directions within the cell:

$$\theta_g = \tan^{-1}\left(\frac{\sin(\theta_s)}{\cos(\theta_s) + \frac{s}{\rho}}\right), \tag{3.17}$$

where ρ is the surface radius, θ_s is the surface zenith angle (with origin at the surface crossing), and s is the distance the particle has travelled from the surface crossing.

Co-centric spherical shell cells (on the z-x plane) represent a useful example to transition from singular angles (e.g. θ) to an angular bins (e.g. $\Delta\theta$). When the $\cos(\Delta\theta_s)$ is positive, particles stream outward from the inner sphere until they intersect the outer sphere. The volume occupied by those particles is an annular cone–a solid of revolution formed by the blue/green region in the left of Figure 3.2 rotated about the z axis. When $\cos(\Delta\theta_s)$ is

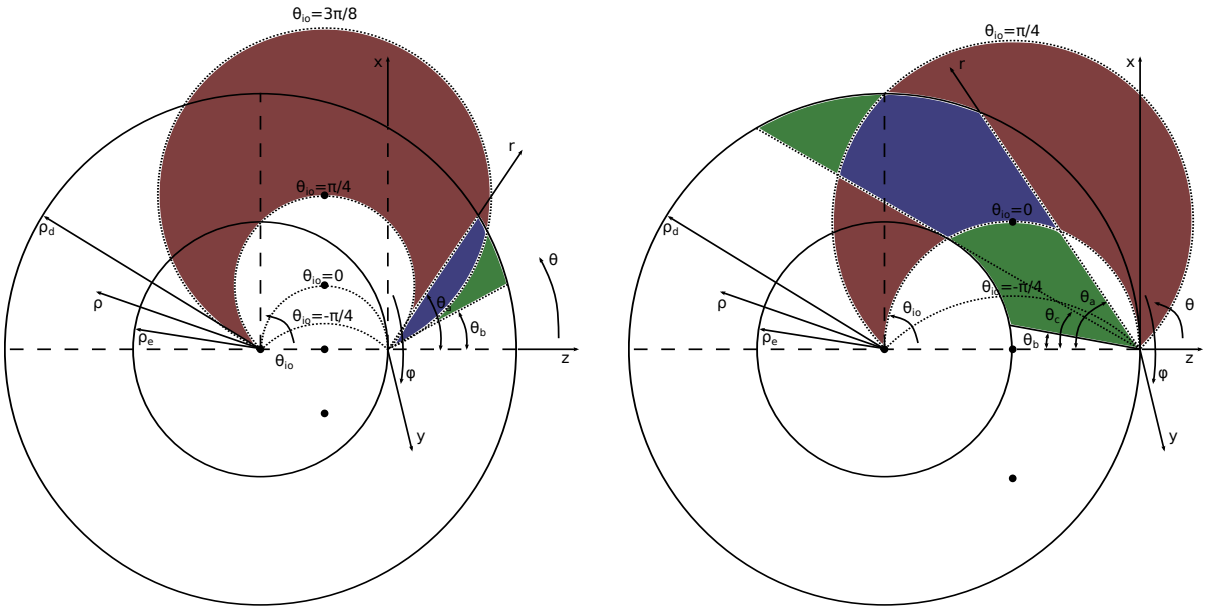

Figure 3.2: Directional smearing for (left) outward- and (right) inward-facing angular bins.

negative, particles stream inward from the outer sphere until they intersect either the inner or outer sphere. The volume occupied by those particles is an annular cone minus the inner sphere and its shadow–a solid of revolution formed by the blue/green region in the right of Figure 3.2 rotated about the z axis.

Whereas the isodirectional curve for a surface angle is a straight line, the isodirectional curve for a global angle is a circle arc that intersects the sphere centers and the surface crossing and has a center located at that global angle. When a particle crosses one of these

curves it possesses the global angle of that curve. Angular bins are the crescent regions between two of the circle arcs–a solid of revolution formed by the blue/red regions in Figure 3.2 rotated about the z axis.

A particle originating in a certain $\Delta\theta_s$ (blue/green) contributes to a certain $\Delta\theta_g$ (blue/red) only when it is within the intersection of the two volumes (blue). Any blue/green region intersects with multiple blue/red regions, so quantities distributed over $\Delta\theta_s$'s contribute to multiple $\Delta\theta_g$'s. The weight of contribution from one blue/green region to a blue region is the volume of the blue region divided by the volume of the blue/green region, so determining how a particles presence is smeared across cell angular bins is essentially reduced to volume integration.

Since all volumes of interest are solids of rotation, the geometry can be further reduced to determining which curves bound a two-dimensional region and where those curves intersect each other. Figure 3.3 shows the blue region from the right of Figure 3.2 divided into four pieces by drawing vertical lines at the curve intersection points. The volume of the

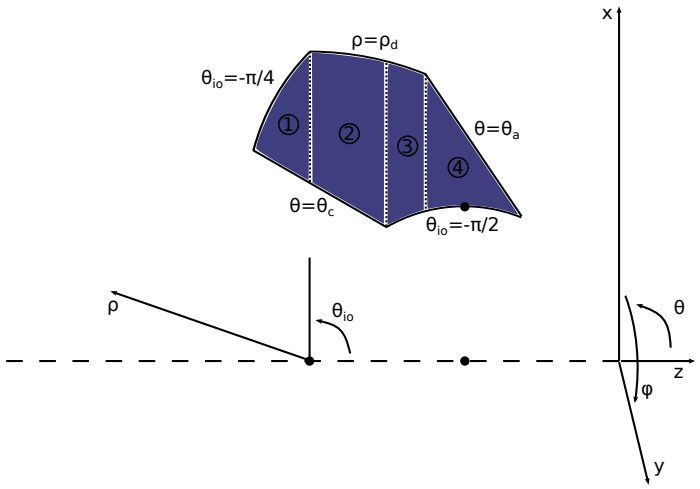

Figure 3.3: The region of intersection, broken into four segments for rotation about the z axis.

blue region is the piecewise sum of four volume integral differences using the disc method [Weisstein, 2011c]:

$$V = \pi \int_a^b dz \ \left| x_{upper}(z)^2 - x_{lower}(z)^2 \right| . \tag{3.18}$$

When a curve is a line segment, its volume of revolution is:

$$V = \frac{\pi}{3} \tan^2(\theta_0) \left[(z - z_0)^3 \right]_a^b , \tag{3.19}$$

where z_0 is the x-intercept, θ_0 is the slope angle, and a and b are the intersection integration

limits. When a curve is a circle arc, its volume of revolution is:

$$V = \pi \left[\left(\rho_0^2 + x_0^2\right) \Delta z \pm x_0 \Delta z \sqrt{\rho_0^2 - \Delta z^2} \pm \rho_0^2 x_0 \tan^{-1}\left(\frac{\Delta z}{\sqrt{\rho_0^2 - \Delta z^2}}\right) - \frac{\Delta z^3}{3} \right]_a^b, \quad (3.20)$$

where $\Delta z \equiv z - z_0$, ρ_0 is the circle radius, x_0 and z_0 are the circle center x-offset and z-offset, and the $\pm$ depends upon whether the arc is the upper or lower portion of the circle.

Intersection points can be organized according to the types of curves by which they are defined. Consistent with Figure 3.2, the curves defined by the global angle bin are i and o, those defined by the surface angle bin are a, b, and c, and those defined by the outer and inner spheres are d and e. For example, the description of the coordinates at the intersection of either i or o and a, b, or c is be denoted as io/abc.

When $\cos(\Delta\theta_s)$ is positive:

$$i/o = \begin{bmatrix} -\rho_i \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad (3.21)$$

$$a/b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad (3.22)$$

$$io/ab = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \cos(\theta_{ab}) \\ \sin(\theta_{ab}) \end{bmatrix} \rho_i \cos(\theta_{ab}) \left(\frac{\tan(\theta_{ab})}{\tan(\theta_{io})} - 1\right), \quad (3.23)$$

$$ab/d = \begin{bmatrix} \cos(\theta_{ab}) \\ \sin(\theta_{ab}) \end{bmatrix} \rho_i \cos(\theta_{ab}) \left(\pm\sqrt{\frac{\rho_o}{\rho_i}\sec^2(\theta_{ab}) - \tan^2(\theta_{ab})} - 1\right), \quad (3.24)$$

$$abio/e = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad (3.25)$$

$$io/d = \begin{bmatrix} \rho_o \frac{\rho_o}{\rho_i} \sin(\theta_{io}) \sin(\theta_{io}) \mp \rho_o \cos(\theta_{io}) \sqrt{1 - \frac{\rho_o^2}{\rho_i^2}\sin^2(\theta_{io})} - \rho_i \\ \rho_o \frac{\rho_o}{\rho_i} \cos(\theta_{io}) \sin(\theta_{io}) \pm \rho_o \sin(\theta_{io}) \sqrt{1 - \frac{\rho_o^2}{\rho_i^2}\sin^2(\theta_{io})} \end{bmatrix}, \quad (3.26)$$

where ρ_i and ρ_o are the inner and outer circle radii, θ_{ab} is the slope angle of either curve a or b, and θ_{io} is the global angle for curve i or o. When $\cos(\Delta\theta_s)$ is negative:

$$i/o = \begin{bmatrix} -\rho_o \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad (3.27)$$

$$a/b = b/c = a/c = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad (3.28)$$

$$io/abc = \left[\begin{array}{c} 0 \\ 0 \end{array} \right], \left[\begin{array}{c} \cos(\theta_{abc}) \\ \sin(\theta_{abc}) \end{array} \right] \rho_o \cos(\theta_{abc}) \left(\frac{\tan(\theta_{abc})}{\tan(\theta_{io})} - 1 \right), \tag{3.29}$$

$$abc/d = \left[\begin{array}{c} 0 \\ 0 \end{array} \right], - \left[\begin{array}{c} \cos(\theta_{abc}) \\ \sin(\theta_{abc}) \end{array} \right] 2\rho_o \cos(\theta_{abc}), \tag{3.30}$$

$$abc/e = \left[\begin{array}{c} 0 \\ 0 \end{array} \right], = - \left[\begin{array}{c} \cos(\theta_{abc}) \\ \sin(\theta_{abc}) \end{array} \right] \rho_o \cos(\theta_{abc}) \left(\pm \sqrt{\frac{\rho_i}{\rho_o} \sec^2(\theta_{abc}) - \tan^2(\theta_{abc})} - 1 \right), \tag{3.31}$$

$$io/d = \left[\begin{array}{c} \cos(\theta_{abc}) \\ \sin(\theta_{abc}) \end{array} \right] \rho_o \cos(\theta_{io}) \left(-1 \mp 1 \right), \tag{3.32}$$

$$io/e = \left[\begin{array}{c} \rho_i \frac{\rho_i}{\rho_o} \sin(\theta_{io}) \sin(\theta_{io}) \mp \rho_i \cos(\theta_{io}) \sqrt{1 - \frac{\rho_i^2}{\rho_o^2} \sin^2(\theta_{io})} - \rho_o \\ \rho_i \frac{\rho_i}{\rho_o} \cos(\theta_{io}) \sin(\theta_{io}) \pm \rho_i \sin(\theta_{io}) \sqrt{1 - \frac{\rho_i^2}{\rho_o^2} \sin^2(\theta_{io})} \end{array} \right], \tag{3.33}$$

where θ_{abc} is the slope angle of either curve a, b, or c and the slope angle for c is defined as:

$$\cos(\theta_c) = -\sqrt{1 - \frac{\rho_i^2}{\rho_o^2}}. \tag{3.34}$$

With the mathematical details of volumes of revolution for both curve types and equations for all intersection coordinates defined, the general algorithm can be stated for the calculation of the directional mapping coefficients:

1. Find all intersection points for a given $\Delta\theta_{ab}$ and $\Delta\theta_{io}$
2. Form $n - 1$ sorted domains from the n real intersection point z coordinates
3. Sample every curve (a, b, c, d, e, i, o) within each domain for x coordinates
4. Determine which curves represent the upper and lower limits of the intersection region
5. If the upper limit curve is above the lower limit curve, calculate the volume difference
6. Repeat for all $[\Delta\theta_{ab}, \Delta\theta_{io}]$ combinations

These coefficients can be used to construct cell angular distributions from surface angular distributions, which are themselves constructed from surface current tallies in Equation 3.16. Alternatively, the intermediate step can be skipped by calculating cell angular distributions

directly from surface current tallies:

$$\alpha_c(E,\vec{\Omega}_i) = \sum_s \sum_j \tau_c(\mu_i,\mu_j) \frac{\frac{F_s^1(E,\mu_i)}{\bar{\mu}_i \ 2\pi\Delta\mu_i}}{\sum_j \frac{F_s^1(E,\mu_j)}{\bar{\mu}_j}}, \tag{3.35}$$

where s is the bounding surface index for cell c and $\tau_c(\mu_i,\mu_j)$ is the angle smearing coefficient for cell c from angle bin μ_i to μ_j. Calculation of the angle smearing coefficients is performed in the `PopulateSphericalDirectionalMappings` method of the `McnpOutputFile` class in `ParseMcnp.py`, the source of which is included below:

Algorithm 3.2: PopulateSphericalDirectionalMappings

```
    def PopulateSphericalDirectionalMappings(self):
        ###
        # Line segment for volumes of revolution
        ###
        class Line:
            def __call__(self, xValue, sign = +1):
                return (xValue - self.GetXOffset()) * self.GetSlope();
            ###
            def __init__(self, xOffset, slope):
                self.xOffset = xOffset;
                self.slope = slope;
                ###
                return;
            ###
            def GetSlope(self):
                return self.slope;
            ###
            def GetXOffset(self):
                return self.xOffset;
            ###
            def RevolutionIndefinite(self, limit, sign):
                return pi / 3. * self.GetSlope() ** 2. * (limit - self.GetXOffset())
                    ** 3.;
            ###
            def Revolve(self, limits, sign = +1):
                return self.RevolutionIndefinite(limits[1], sign) - self.
                    RevolutionIndefinite(limits[0], sign);
        ###
        # Circle arc for volumes of revolution
        ###
        class Circle(Line):
            def __call__(self, xValue, sign = +1):
                return self.GetYOffset() + sign * (self.GetRadius() ** 2. - (xValue -
                    self.GetXOffset()) ** 2.) ** 0.5;
            ###
            def __init__(self, xOffset, yOffset, radius):
                self.xOffset = xOffset;
                self.yOffset = yOffset;
                self.radius = radius;
                ###
                return;
            ###
            def GetRadius(self):
                return self.radius;
```

```
42              ###
43              def GetYOffset(self):
44                  return self.yOffset;
45              ###
46              def RevolutionIndefinite(self, limit, sign):
47                  radical = self.GetRadius() ** 2. - (limit - self.GetXOffset()) ** 2.;
48                  radical = [radical, 0][abs(radical) < epsilon];
49                  return pi * ((self.GetRadius() ** 2. + self.GetYOffset() ** 2.) * (
                        limit - self.GetXOffset()) + sign * self.GetYOffset() * (limit -
                        self.GetXOffset()) * radical ** 0.5 - sign * self.GetRadius() **
                        2. * self.GetYOffset() * ArcTangent2(self.GetXOffset() - limit,
                        radical ** 0.5) - (limit - self.GetXOffset()) ** 3. / 3.);
50          ###
51          # Boundaries confining overlapping regions
52          ###
53          def Parameters2Boundaries(radiusInner, radiusOuter, positiveMu, curves,
                domains, withIO = True):
54              domain2Boundaries = {};
55              glance = (radiusInner ** 2. - radiusOuter ** 2.) / radiusOuter;
56              ###
57              for domain in domains:
58                  midpoint = sum(domain) / len(domain);
59                  lowerCeiling = {(key, sign) : value for key, sign in (('i', -1), ('o',
                        +1), ('a', +1), ('d', +1)) if key in curves for value in [curves[
                        key](midpoint, sign)] if not isinstance(value, complex)};
60                  upperFloor = {(key, sign) : value for key, sign in (('i', +1), ('o',
                        -1), ('b', -1), ('c', -1), ('d', -1), ('e', +1)) if key in curves
                        for value in [curves[key](midpoint, sign)] if not isinstance(value
                        , complex)};
61                  ###
62                  if not positiveMu and midpoint > glance:
63                      try:
64                          upperFloor.pop(('c', -1));
65                      except KeyError:
66                          pass;
67                  ###
68                  upperCeiling = lowerCeiling.copy();
69                  try:
70                      upperCeiling.pop(('i', -1));
71                  except KeyError:
72                      pass;
73                  ###
74                  lowerFloor = upperFloor.copy();
75                  try:
76                      lowerFloor.pop(('i', +1));
77                  except KeyError:
78                      pass;
79                  ###
80                  if withIO:
81                      if all(key not in dictionary for key in (('o', +1), ('o', -1)) for
                            dictionary in (lowerCeiling, lowerFloor, upperCeiling,
                            upperFloor)) and 'o' in curves:
82                          continue;
83                      else:
84                          for dictionary in (lowerCeiling, lowerFloor, upperCeiling,
                                upperFloor):
85                              for key in (('o', +1), ('o', -1), ('i', +1), ('i', -1)):
86                                  try:
87                                      dictionary.pop(key);
88                                  except KeyError:
```

```
89                                  pass;
90                          ###
91                          extrema = [(FindExtremeKey(lowerCeiling, min), FindExtremeKey(
                                lowerFloor, max)), (FindExtremeKey(upperCeiling, min),
                                FindExtremeKey(upperFloor, max))];
92                          ###
93                          # Filter out non-overlapping and repeated ranges
94                          ###
95                          boundaries = {(uKey, uSign, lKey, lSign) for (uKey, uSign), (lKey,
                                lSign) in extrema if curves[uKey](midpoint, uSign) > curves[lKey](
                                midpoint, lSign)};
96                          if boundaries:
97                              domain2Boundaries[domain] = boundaries;
98                      ###
99                      return domain2Boundaries;
100                 ###
101                 # Find extreme key
102                 ###
103                 def FindExtremeKey(dictionary, extreme):
104                     extremum = extreme(dictionary.values());
105                     for key in dictionary:
106                         if dictionary[key] == extremum:
107                             return key;
108                 ###
109                 # Revolved volume between two curves
110                 ###
111                 def Parameters2Integral(curves, domain2Boundaries):
112                     return sum(curves[upperKey].Revolve(domain, upperSign) - curves[lowerKey].
                            Revolve(domain, lowerSign) for domain, boundaries in sorted(
                            domain2Boundaries.items()) for upperKey, upperSign, lowerKey,
                            lowerSign in boundaries);
113                 ###
114                 # Spherical surface directional smearing
115                 ###
116                 def AngleSmearingCoefficients(radiusInner, radiusOuter, angles):
117                     radiusInner += epsilon * radiusOuter * (0 == radiusInner);
118                     ###
119                     anglePairs = [(-1., angles[0]), ];
120                     anglePairs.extend((angles[index], angles[index + 1]) for index in range(
                            len(angles) - 1));
121                     ###
122                     angles2Coefficient = {};
123                     for cosA, cosB in anglePairs:
124                         positiveMu = cosA >= 0 and cosB >= 0;
125                         if positiveMu:
126                             radiusOffset = radiusInner;
127                         else:
128                             cosB, cosA = cosA, cosB;
129                             radiusOffset = radiusOuter;
130                         ###
131                         for cosI, cosO in anglePairs:
132                             ###
133                             # Curves
134                             ###
135                             curves = {};
136                             ###
137                             # AB(C) Lines
138                             ###
139                             cosABCs = [('a', cosA), ('b', cosB)];
140                             if not positiveMu:
```

```
141                         cosC = -(1. - (radiusInner / radiusOuter) ** 2.) ** 0.5;
142                         cosABCs.append(('c', cosC));
143                     ###
144                     for key, cosABC in cosABCs:
145                         if 0. != cosABC:
146                             tanABC = (1. - cosABC ** 2.) ** 0.5 / cosABC;
147                             curves[key] = Line(xOffset = 0, slope = tanABC);
148                     ###
149                     # IO Circles
150                     ###
151                     for key, cosIO in [('i', cosI), ('o', cosO)]:
152                         if 1. == abs(cosIO):
153                             continue;
154                         ###
155                         sinIO = (1. - cosIO ** 2.) ** 0.5;
156                         curves[key] = Circle(xOffset = -radiusOffset / 2, yOffset =
                                radiusOffset / 2 * cosIO / sinIO, radius = radiusOffset /
                                2 / sinIO);
157                     ###
158                     # DE Circles
159                     ###
160                     curves['d'] = Circle(xOffset = -radiusOffset, yOffset = 0, radius
                            = radiusOuter);
161                     curves['e'] = Circle(xOffset = -radiusOffset, yOffset = 0, radius
                            = radiusInner);
162                     ###
163                     # Domains
164                     ###
165                     intersections = [];
166                     ###
167                     io = [-radiusOffset, 0];
168                     ab = abc = [0];
169                     ioab = ioabc = [0];
170                     abd = abcd = abce = [];
171                     ###
172                     cosABCs = [cosA, cosB];
173                     if not positiveMu:
174                         cosC = -(1. - (radiusInner / radiusOuter) ** 2.) ** 0.5;
175                         cosABCs.append(cosC);
176                     for cosABC in cosABCs:
177                         if 0 == cosABC:
178                             if positiveMu:
179                                 abd.extend((-radiusInner - radiusOuter, -radiusInner +
                                        radiusOuter));
180                             else:
181                                 abcd.extend((-radiusOuter - radiusOuter, -radiusOuter
                                        + radiusOuter));
182                                 abce.extend((-radiusOuter - radiusInner, -radiusOuter
                                        + radiusInner));
183                         else:
184                             tanABC = (1. - cosABC ** 2.) ** 0.5 / cosABC;
185                             if positiveMu:
186                                 abd.extend(radiusInner * cosABC ** 2. * (sign * ((
                                        radiusOuter / radiusInner / cosABC) ** 2. - tanABC
                                        ** 2) ** 0.5 - 1.) for sign in (+1, -1));
187                             else:
188                                 abcd.append(-2 * radiusOuter * cosABC ** 2.);
189                                 radical = (radiusInner / radiusOuter / cosABC) ** 2. -
                                        tanABC ** 2.;
190                                 radical = [radical, 0][abs(radical) < epsilon];
```

```
191                             abce.extend(radiusOuter * cosABC ** 2. * (sign *
                                    radical ** 0.5 - 1) for sign in (+1, -1));
192                         ###
193                         for cosIO in (cosI, cosO):
194                             if 1. == abs(cosIO):
195                                 continue;
196                             ###
197                             cotIO = cosIO / (1. - cosIO ** 2.) ** 0.5;
198                             intersection = cosABC ** 2. * radiusOffset * (tanABC *
                                     cotIO - 1.);
199                             if positiveMu:
200                                 ioab.append(intersection);
201                             else:
202                                 ioabc.append(intersection);
203                     ###
204                     if positiveMu:
205                         iod = [radiusOuter - radiusInner];
206                         ioe = [];
207                         extrema = [0, radiusOuter - radiusInner];
208                     else:
209                         iod = [0];
210                         ioe = [-radiusOuter + radiusInner];
211                         extrema = [-radiusOuter - radiusOuter, 0];
212                     ###
213                     for cosIO in (cosI, cosO):
214                         if 1. == abs(cosIO):
215                             continue;
216                         ###
217                         sinIO = (1. - cosIO ** 2.) ** 0.5;
218                         io.extend(radiusOffset / 2. * ( -sign / sinIO - 1.) for sign
                                in (+1, -1))
219                         if positiveMu:
220                             iod.extend((radiusOuter * sinIO) ** 2. / radiusInner -
                                    sign * radiusOuter * cosIO * (1. - (radiusOuter *
                                    sinIO / radiusInner) ** 2.) ** 0.5 - radiusInner for
                                    sign in (+1, -1));
221                         else:
222                             iod.extend(cosIO ** 2. * radiusOuter * (-1 - sign) for
                                    sign in (+1, -1));
223                             ioe.extend((radiusInner * sinIO) ** 2. / radiusOuter -
                                    sign * radiusInner * cosIO * (1. - (radiusInner *
                                    sinIO / radiusOuter) ** 2.) ** 0.5 - radiusOuter for
                                    sign in (+1, -1));
224                     ###
225                     intersections = sorted({intersection for intersections in [(io,
                            abc, ioabc, abcd, abce, iod, ioe), (io, ab, ioab, abd, iod)][
                            positiveMu] for intersection in intersections if not
                            isinstance(intersection, complex) if extrema[0] <=
                            intersection <= extrema[1]});
226                     ###
227                     domains = [(intersections[index], intersections[index + 1]) for
                            index in range(len(intersections) - 1)];
228                     ###
229                     # Boundaries
230                     ###
231                     part = Parameters2Boundaries(radiusInner, radiusOuter, positiveMu,
                             curves, domains);
232                     whole = Parameters2Boundaries(radiusInner, radiusOuter, positiveMu
                            , curves, domains, False);
233                     ###
```

```
234                     # Integrals
235                     ###
236                     index = (max(cosA, cosB), max(cosI, cosO));
237                     if whole:
238                         angles2Coefficient[index] = Parameters2Integral(curves, part)
                                / Parameters2Integral(curves, whole);
239                     else:
240                         angles2Coefficient[index] = 0.;
241             ###
242             return angles2Coefficient;
243         ###
244         sphericalSurfaceNumbers = sorted(set(surface.GetNumber() for cell in self.
                GetCells() for rootCell in self.FindRootCells(cell.GetNumber()) for
                surface in self.FindCellSurfaces(rootCell.GetNumber()) if surface.
                GetIsSphere()));
245         ###
246         numberOfAngleBins = senseFile.GetParameter('numberOfAngleBins');
247         angleBins = list(-1. + 2. * (index + 1) / numberOfAngleBins for index in range
                (int(numberOfAngleBins)));
248         cellNumber2Surfaces = {rootCell.GetNumber() : [surface for surface in self.
                FindCellSurfaces(rootCell.GetNumber()) if surface.GetNumber() in
                sphericalSurfaceNumbers] for cell in self.GetCells() for rootCell in self.
                FindRootCells(cell.GetNumber())};
249         cellNumber2Radii = {};
250         for cellNumber, surfaces in cellNumber2Surfaces.items():
251             radii = [surface.GetDimension() for surface in surfaces];
252             if 1 == len(surfaces):
253                 surfaceNumber = [surfaceNumber for surfaceNumber in self.FindCell(
                        cellNumber).GetSurfaceNumbers() if abs(surfaceNumber) in
                        sphericalSurfaceNumbers][0];
254                 if 0 > surfaceNumber:
255                     radii.append(0.);
256                 else:
257                     continue;
258             cellNumber2Radii[cellNumber] = sorted(radii);
259         ###
260         radii2Angles2Coefficient = {(radiusInner, radiusOuter) :
                AngleSmearingCoefficients(radiusInner, radiusOuter, angleBins) for
                radiusInner, radiusOuter in {tuple(radii) for radii in cellNumber2Radii.
                values()}};
261         self.cellNumber2AngleSmearing = {cellNumber : radii2Angles2Coefficient[(
                radiusInner, radiusOuter)] for cellNumber, (radiusInner, radiusOuter) in
                cellNumber2Radii.items()};
262         ###
263         return;
```

These coefficients are then used to calculate cell angular distributions (Equation 3.35) in the `GetCellAngularDistributionEnergyDistribution` method of the `McnpOutputFile` class in `ParseMcnp.py`, the source of which is shown below:

Algorithm 3.3: GetCellAngularDistributionEnergyDistribution

```
1     def GetCellAngularDistributionEnergyDistribution(self, cellNumber, particles = 'np
          '):
2         ###
3         # Accumulate angular fluxes from each surface (\psi * A * d\Omega)
4         ###
5         surfaceNumbers = self.FindCell(cellNumber).GetSurfaceNumbers();
6         angle2AngularFlux = {};
```

```
7           for surface in self.FindCellSurfaces(cellNumber):
8               ###
9               # Extract surface sense
10              ###
11              for surfaceSense in (+1, -1):
12                  if surface.GetNumber() * surfaceSense in surfaceNumbers:
13                      break;
14              ###
15              for tally in self.GetTallys('f1'):
16                  ###
17                  # Kick out tallies that do not contain surface
18                  ###
19                  if surface not in tally:
20                      continue;
21                  ###
22                  # Kick out tallies that do not contain particles
23                  ###
24                  if particles not in tally:
25                      continue;
26                  ###
27                  for fromAngle in tally.GetAngles():
28                      ###
29                      # Flip surface angles that depart cell
30                      ###
31                      flipSign = surfaceSense * (-1) ** (fromAngle <= 0);
32                      ###
33                      for toAngle in tally.GetAngles():
34                          try:
35                              angleSmearing = self.GetCellNumberAngleSmearing(cellNumber
                                    , (flipSign * fromAngle, toAngle));
36                          except KeyError:
37                              angleSmearing = flipSign * fromAngle == toAngle;
38                          ###
39                          try:
40                              angle2AngularFlux[toAngle] += angleSmearing * SafeDivide(
                                    tally.GetEnergyDistribution(surface.GetNumber(),
                                    fromAngle), abs(tally.GetAngleMean(fromAngle)));
41                          except KeyError:
42                              angle2AngularFlux[toAngle] = angleSmearing * SafeDivide(
                                    tally.GetEnergyDistribution(surface.GetNumber(),
                                    fromAngle), abs(tally.GetAngleMean(fromAngle)));
43          ###
44          # Construct scalar flux (\phi * A) from accumulated angular fluxes (\psi * A *
                d\Omega)
45          ###
46          scalarFlux = sum(angle2AngularFlux.values());
47          ###
48          # Construct angular distribution from angular flux and scalar flux
49          ###
50          return {angle : SafeDivide(angularFlux, scalarFlux * tally.GetAngleWidth(angle
                )) for angle, angularFlux in angle2AngularFlux.items()};
```

In the following section, angular segregation and discretization are achieved for neutron sinks and sources as they were for adjoint distributions.

3.2.4 Angular segregation and discretization of neutron sinks and sources

Depending on the nature of the input parameter p, $\Delta\mathbb{A}$ (or $p\frac{\partial\mathbb{A}}{\partial p}$) can correspond to a neutron sink mechanism, neutron source mechanism, or both–essentially the second and/or third terms in Equation 2.4. For neutron sinks, $\Delta\mathbb{A}\psi$ is the distribution rate of neutrons preceding a reaction:

$$\Delta\mathbb{A}_{sink}\psi \equiv \Sigma_{sink}(\vec{r}, E)\ \psi(\vec{r}, E, \vec{\Omega}) = n_{sink}(\vec{r}, E, \vec{\Omega}); \tag{3.36}$$

for neutron sources, $\Delta\mathbb{A}\psi$ is the distribution rate of neutrons following a reaction:

$$\begin{aligned}\Delta\mathbb{A}_{source}\psi \equiv \int_0^\infty dE' \int_{4\pi} d\vec{\Omega}'\ \nu_{source}(E')\ \Sigma_{source}(\vec{r}, E' \to E, \vec{\Omega}' \to \vec{\Omega})\ \psi(\vec{r}, E', \vec{\Omega}') \\ = n_{source}(\vec{r}, E, \vec{\Omega}).\end{aligned} \tag{3.37}$$

For both cases, the same procedure can be used on $\Delta\mathbb{A}\psi$ to segregate angular dependence from the rest:

$$\begin{aligned}\psi^\dagger\Delta\mathbb{A}\psi &= \phi^\dagger(\vec{r}, E) \int_{\vec{\Omega}} d\vec{\Omega}\ \alpha^\dagger(\vec{r}, E, \vec{\Omega})\ n_x(\vec{r}, E, \vec{\Omega}) \\ &= \phi^\dagger(\vec{r}, E)\ \frac{\int_{\vec{\Omega}} d\vec{\Omega}\ n_x(\vec{r}, E, \vec{\Omega})}{\int_{\vec{\Omega}} d\vec{\Omega}\ n_x(\vec{r}, E, \vec{\Omega})} \int_{\vec{\Omega}} d\vec{\Omega}\ \alpha^\dagger(\vec{r}, E, \vec{\Omega})\ n_x(\vec{r}, E, \vec{\Omega}) \\ &= \phi^\dagger(\vec{r}, E)\ \frac{n_x(\vec{r}, E)}{n_x(\vec{r}, E)} \int_{\vec{\Omega}} d\vec{\Omega}\ \alpha^\dagger(\vec{r}, E, \vec{\Omega})\ n_x(\vec{r}, E, \vec{\Omega}) \\ &= \phi^\dagger(\vec{r}, E)\ n_x(\vec{r}, E) \int_{\vec{\Omega}} d\vec{\Omega}\ \alpha^\dagger(\vec{r}, E, \vec{\Omega})\ \frac{n_x(\vec{r}, E, \vec{\Omega})}{n_x(\vec{r}, E)},\end{aligned} \tag{3.38}$$

where n_x is either the neutron source or sink distribution.

When the neutron sink distribution rate Equation 3.36 is examined, it is clear that angular dependence rests entirely within ψ. Therefore, the fraction in Equation 3.38 is physically equivalent to Equation 3.4 for neutron sinks. The neutron sink distribution rate outside of the directional integral in Equation 3.38 is equal to the neutron sink reaction rate ; the rate at which neutrons are lost is equal to the rate at which they incur sink reactions. Therefore, for neutron sinks, the fraction can be replaced with α and $n_{sink}(\vec{r}, E)$ can be

replaced with the linear functional reaction rate $R_{sink}(\vec{r}, E)$:

$$\psi^{\dagger}\Delta\mathbb{A}_{sink}\psi = \phi^{\dagger}(\vec{r}, E)\ R_{sink}(\vec{r}, E) \int\limits_{\vec{\Omega}} d\vec{\Omega}\ \alpha^{\dagger}(\vec{r}, E, \vec{\Omega})\ \alpha(\vec{r}, E, \vec{\Omega}). \tag{3.39}$$

When Equation 3.38 is considered for neutron sources, the realization is made that the neutron source distribution rate is related to the source reaction rate , but not in a straightforward manner. With most fissions, thermal neutrons beget multiple fast neutrons, distributed in a fission energy spectrum. $(n, 2n)$ and $(n, 3n)$ always birth two or three neutrons respectively at lower kinetic energies. Scattering releases one neutron at a lower kinetic energy, except within thermal energies, where the neutron may be energized. F^{M4} cell flux multiplier tallies identify the distribution rate of neutrons at the onset of reactions, but know nothing of the neutrons after the reaction occurs. Tally tagging, a new feature in MCNP6, is uniquely suited to provide post-reaction information.

Any particle is able to contribute to a tally for a neutron sink, but by restricting a tally to only those particles that are born in a manner corresponding to p, a neutron source could be extracted. Tally tagging can be considered as a form of particle-wise filtering; it allows for binning of tallies according to how particles are tagged and a particle's tag is determined by how it is born (some combination of the cell, isotope, and reaction type) [Pelowitz *et al.*, 2011]. All that remains is to form a tallying scheme that efficiently extracts neutron source rate distributions.

As depicted in Figure 3.4, all neutrons born within a cell have one of two ultimate fates: to collide within the birth cell (paths b and d), or to escape from that cell before colliding (paths a, c, and e). The rate of occurrence of the first fate can be extracted with an F^{M4} cell

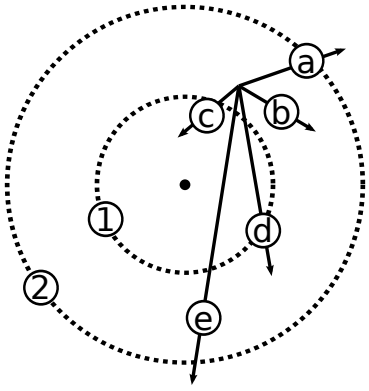

Figure 3.4: The five distinct paths for a particle born in a spherical shell until collision.

flux multiplier tally for collision reactions ($MT = 1$). The rate of the second can be counted by using F^1 surface current tallies to tabulate net surface crossings. The complication of

reentrant cells can be overcome with angular binning. For example, referring to Figure 3.4 again, if surfaces 1 and 2 have inward/outward angular binning (with respect to the birth cell), paths are tallied as in Table 3.1. By subtracting inward surface crossing from outward

Surface #	**Sense**	**Paths**
1	inward	$d+e$
1	outward	$c+d+e$
2	inward	$\ldots$
2	outward	$a+e$

Table 3.1: Particle path contributions to directional surface crossing tallies.

crossings, net leakages can be obtained:

$$F^1_{1,out} - F^1_{1,in} + F^1_{2,out} - F^1_{2,in} = (c+d+e) - (d+e) + (a+e) - () = (a+c+e). \quad (3.40)$$

This expression can be generalized for a cell c bounded by any number of surfaces s, reentrant or not (even multiply so):

$$\text{net leakage} = \sum_s \sum_\mu \left((-1)^{(\Delta\mu \text{ is inward})} \; F^1_{s,\mu} \right). \quad (3.41)$$

Angular distributions are extracted in the usual manner but with tally tags. With schemes defined for extracting neutron source rates and distributions, the segregated bilinear functional (Equation 3.38) can be written for neutron sources:

$$\psi^\dagger \Delta\mathbb{A}_{source}\psi = \phi^\dagger(\vec{r},E)\; n_{source}(\vec{r},E) \int\limits_{\vec{\Omega}} d\vec{\Omega}\; \alpha^\dagger(\vec{r},E,\vec{\Omega})\; \alpha_{source}(\vec{r},E,\vec{\Omega}). \quad (3.42)$$

The expressions for neutron sinks and sources (Equations 3.39 and 3.42) can now be discretized, written entirely in terms of standard MCNP6 tallies. As before, cell fluxes and cell reaction rate densities can be replaced with F^4 cell flux tallies and F^{M4} cell flux multiplier tallies, respectively:

$$\psi^\dagger \Delta\mathbb{A}_{sink}\psi \cong F^{4\dagger}_c(E)\; F^{M4}_{c,z,x}(E) \sum_i \Delta\mu_i\; \alpha^\dagger_c(E,\mu_i)\; \alpha_c(E,\mu_i), \quad (3.43)$$

$$\psi^\dagger \Delta\mathbb{A}_{source}\psi \cong F^{4\dagger}_c(E)\; n_{c,source}(E) \sum_i \Delta\mu_i\; \alpha^\dagger_c(E,\mu_i)\; \alpha_{c,source}(E,\mu_i), \quad (3.44)$$

where c is the cell index, i is the discrete angular bin index, $n_{c,source}(E)$ is the neutron source

rate, defined as:

$$n_{c,source}(E) = F^{M4}_{c,z=*,x=1}(E) + \sum_{s}\sum_{\mu}\left((-1)^{(\Delta\mu \text{ is inward})}\ F^{1}_{s,\mu}\right), \tag{3.45}$$

with directional mapping taken into account, and all cell angular distributions α_c are defined as in Equation 3.35. It won't be shown until Section 3.5, but for now it is taken for granted that adjoint distributions can be extracted in the same manner as forward flux, within an adjoint transport calculation. Every bilinear functional product is performed externally to MCNP6 over discrete regions of phase space, in which physical quantities are averaged over cells, angles, and energy bins. While the treatment is not exact, the discrete summation over all phase space approaches the continuous integral as the bin size is shrunk to arbitrarily small size.

The neutron source surface accumulation scheme with tally tagging for both source rate and angular distribution is demonstrated in the `GetCellSourceEnergyAngularDistribution` method of the `McnpOutputFile` class within `ParseMcnp.py`:

Algorithm 3.4: GetCellSourceEnergyAngularDistribution

```
1   def GetCellSourceEnergyAngularDistribution(self, cellNumber, materialNumber, za,
        reactionNumber, particles = 'np'):
2       ###
3       # Accumulate all possible source particle tracks
4       # Particles can originate from within any sister leaf cell
5       ###
6       surfaceCrossingAngularDistribution = {};
7       collisionReactionNumber = 1;
8       collisions = 0.;
9       ###
10      for rootCell in self.FindRootCells(cellNumber):
11          ###
12          # Extract root cell surface numbers for sense
13          ###
14          rootCellSurfaceNumbers = rootCell.GetSurfaceNumbers();
15          for sisterCell in self.FindLeafCells(rootCell):
16              ###
17              # Build tally tag
18              ###
19              tallyTag = (1e5 * sisterCell.GetNumber()) + za + (1e-5 *
                    reactionNumber);
20              ###
21              # Accumulate root cell bounding surface crossings
22              # Root cell bounding surface crossings record particles that collide
                    outside of the root cell
23              ###
24              for surface in self.FindCellSurfaces(rootCell.GetNumber()):
25                  ###
26                  # Kick out transformed surfaces
27                  # This may be dangerous, but was the best conceived way to
                        eliminate the laser port surfaces
28                  ###
29                  if surface.GetTransformationNumber():
30                      continue;
31                  ###
```

```
32                      # Extract surface sense
33                      ###
34                      for surfaceSense in (+1, -1):
35                          if surface.GetNumber() * surfaceSense in
                                rootCellSurfaceNumbers:
36                              break;
37                      ###
38                      # Iterate over F1 tallies
39                      ###
40                      for tally in self.GetTallys('f1'):
41                          ###
42                          # Kick out tallies that do not contain surface
43                          ###
44                          if surface not in tally:
45                              continue;
46                          ###
47                          # Kick out tallies that do not contain particles
48                          ###
49                          if particles not in tally:
50                              continue;
51                          ###
52                          # Kick out tallies that do not contain tally tag
53                          ###
54                          if tallyTag not in tally:
55                              continue;
56                          ###
57                          # Preserve surface crossing tally for later
58                          ###
59                          surfaceCrossingTally = tally;
60                          for fromAngle in tally.GetAngles():
61                              ###
62                              # Flip surface angles that depart cell
63                              ###
64                              flipSign = surfaceSense * (-1) ** (fromAngle <= 0);
65                              ###
66                              for toAngle in tally.GetAngles():
67                                  try:
68                                      angleSmearing = self.GetCellNumberAngleSmearing(
                                            cellNumber, (flipSign * fromAngle, toAngle));
69                                  except:
70                                      angleSmearing = flipSign * fromAngle == toAngle;
71                                  ###
72                                  try:
73                                      surfaceCrossingAngularDistribution[toAngle] += -
                                            flipSign * angleSmearing * tally.
                                            GetEnergyDistribution(surface.GetNumber(),
                                            tallyTag, fromAngle);
74                                  except KeyError:
75                                      surfaceCrossingAngularDistribution[toAngle] = -
                                            flipSign * angleSmearing * tally.
                                            GetEnergyDistribution(surface.GetNumber(),
                                            tallyTag, fromAngle);
76                  ###
77                  # Accumulate collisions within sister leaf cells
78                  # Sister leaf cell collisions record particles that collide within the
                        root cell
79                  ###
80                  sisterCellMaterialNumber = sisterCell.GetMaterialNumber();
81                  ###
82                  # Divide by number of physical instantiations of sister cell
```

```
83                  ###
84                  divisor = 1.;
85                  if self.GetIsMultiplyRootedLeafCell(sisterCell.GetNumber()):
86                      divisor *= len(self.GetMultiplyRootedLeafCell(sisterCell.GetNumber
                            ()));
87                  ###
88                  # Iterate over FM4 tallies
89                  ###
90                  for tally in self.GetTallys('fm4'):
91                      ###
92                      # Kick out tallies that do not contain sister cell
93                      ###
94                      if sisterCell not in tally:
95                          continue;
96                      ###
97                      # Kick out tallies that do not contain particles
98                      ###
99                      if particles not in tally:
100                         continue;
101                     ###
102                     # Kick out tallies that do not contain tally tag
103                     ###
104                     if tallyTag not in tally:
105                         continue;
106                     ###
107                     for multiplierBin in tally.GetMultiplierBins():
108                         ###
109                         # Divide out non-unity multipliers
110                         ###
111                         multiplier = SafeDivide(sisterCell.GetVolume(), abs(
                                multiplierBin[1]));
112                         ###
113                         collisions += SafeDivide(tally.GetEnergyDistribution(
                                sisterCell.GetNumber(), tallyTag, (
                                sisterCellMaterialNumber, collisionReactionNumber)) *
                                multiplier, divisor);
114             ###
115             # Construct source neutron rate
116             ###
117             sourceRate = sum(surfaceCrossingAngularDistribution.values()) + collisions;
118             ###
119             # Construct source neutron angular distribution
120             ###
121             angle2AngularFlux = {angle : SafeDivide(surfaceCrossings, abs(
                    surfaceCrossingTally.GetAngleMean(angle))) for angle, surfaceCrossings in
                    surfaceCrossingAngularDistribution.items()};
122             scalarFlux = sum(angle2AngularFlux.values());
123             angularDistribution = {angle : SafeDivide(angularFlux, scalarFlux *
                    surfaceCrossingTally.GetAngleWidth(angle)) for angle, angularFlux in
                    angle2AngularFlux.items()};
124             ###
125             return sourceRate, angularDistribution;
```

Bilinear functionals are constructed and then normalized within the `CalculateResponseImplicitSensitivitys` method of the `SensitivityUncertaintyCalculator` class in `ParseMcnp.py` in a manner similar to Algorithm 3.1:

Algorithm 3.5: CalculateResponseImplicitSensitivitys

```
1    def CalculateResponseImplicitSensitivitys(self, response, sumOverCells, sumOverZas
         , sumOverReactions):
2        forwardMcnp = self.GetForwardMcnp();
3        ###
4        # Extract response portions
5        ###
6        dividend = response.GetDividend();
7        divisor = response.GetDivisor();
8        ###
9        # Extract total response
10       ###
11       totalResponse = forwardMcnp.GetTotal(dividend);
12       signs = [+1];
13       if divisor:
14           totalResponse *= SafeDivide(1, forwardMcnp.GetTotal(divisor));
15           signs.append(-1);
16       ###
17       # Kick out responses for which either the dividend or divisor are zero-valued
18       ###
19       if not totalResponse:
20           return {};
21       ###
22       # Perturbation
23       ###
24       perturbation = response.GetPerturbation();
25       isPerturbation = bool(perturbation);
26       if isPerturbation:
27           perturbCellNumber, perturbZa, perturbReactionNumber = perturbation[0];
28       ###
29       # cell number --> importance
30       # Non-dimensional importance is equal to the adjoint flux multiplied by the
             volume, or the adjoint particle track-length
31       # cell number --> adjoint angular distribution
32       ###
33       cellNumber2Importance = {};
34       cellNumber2AdjointAngular = {};
35       for sign in signs:
36           adjointMcnp = ReadMcnpFile('{}.o'.format(response.GetFileName('{}-adj'.
                 format(forwardMcnp.GetFileName().replace('.o', ''))), sign)));
37           for cellNumber in adjointMcnp.GetTallyIndices('f4'):
38               try:
39                   cellNumber2Importance[cellNumber] += sign * adjointMcnp.
                         GetCellImportancePerResponseEnergyDistribution(cellNumber);
40                   cellNumber2AdjointAngular[cellNumber] = adjointMcnp.
                         GetCellAngularDistributionEnergyDistribution(cellNumber);
41               except KeyError:
42                   cellNumber2Importance[cellNumber] = sign * adjointMcnp.
                         GetCellImportancePerResponseEnergyDistribution(cellNumber);
43                   cellNumber2AdjointAngular[cellNumber] = adjointMcnp.
                         GetCellAngularDistributionEnergyDistribution(cellNumber);
44       ###
45       # Iterate over reaction rate tallys
46       ###
47       key2ImplicitSensitivity = {};
48       for cellNumber, multiplierBins in sorted(forwardMcnp.GetTallyIndices('fm4').
             items()):
49           ###
50           # Kick out non-rooted cells
51           # Only root cells have meaningful angular distributions
52           ###
```

```
53                  if forwardMcnp.FindCell(cellNumber).GetUniverse():
54                      continue;
55                  ###
56                  # Cell importance energy distribution
57                  ###
58                  importance = cellNumber2Importance[cellNumber];
59                  ###
60                  # Forward neutron sink angular distribution
61                  ###
62                  sinkAngular = forwardMcnp.GetCellAngularDistributionEnergyDistribution(
                        cellNumber);
63                  ###
64                  # Sink angular integral
65                  ###
66                  sinkAngularIntegral = self.CalculateAngularIntegral(
                        cellNumber2AdjointAngular[cellNumber], sinkAngular);
67                  ###
68                  # Iterate over multiplier bins
69                  ###
70                  for materialNumber, reactionNumber in multiplierBins:
71                      try:
72                          ###
73                          # Extract single-za from material number
74                          ###
75                          za = forwardMcnp.GetMaterialNumber2SingleZa()[materialNumber];
76                      except KeyError:
77                          ###
78                          # Kick out non-single-za material numbers
79                          ###
80                          continue;
81                      ###
82                      # Kick out if indices don't match perturbation indices
83                      ###
84                      if isPerturbation:
85                          # c
86                          if perturbCellNumber not in (None, cellNumber):
87                              continue;
88                          # z
89                          if perturbZa not in (None, za):
90                              continue;
91                          # x
92                          if perturbReactionNumber not in (None, reactionNumber):
93                              continue;
94                      ###
95                      # Neutron sink
96                      ###
97                      sinkRate = forwardMcnp.GetCellSinkEnergyDistribution(cellNumber,
                            materialNumber, reactionNumber);
98                      ###
99                      # Kick out zero-valued neutron sinks
100                     # If the neutron sink is zero, the neutron source certainly is, too
101                     ###
102                     if not float(sinkRate):
103                         continue;
104                     ###
105                     # Neutron source
106                     ###
107                     sourceRate, sourceAngular = 0., 0.;
108                     if reactionNumber in implicitReactionNumbers and
                            implicitReactionNumbers[reactionNumber]:
```

```
109                     sourceRate, sourceAngular = forwardMcnp.
                            GetCellSourceEnergyAngularDistribution(cellNumber,
                            materialNumber, za, reactionNumber);
110                 ###
111                 # Source angular integral
112                 ###
113                 sourceAngularIntegral = self.CalculateAngularIntegral(
                        cellNumber2AdjointAngular[cellNumber], sourceAngular);
114                 ###
115                 # Calculate implicit sensitivity
116                 # The implicit sensitivity due to source neutrons belongs to the
                        energy bins that caused them ##### For MT=4, 16, 17 this is fine
                        ... for MT=18, we need to weight by nu(E').
117                 ###
118                 implicitSensitivity = importance * sinkRate * sinkAngularIntegral;
119                 implicitSensitivity += SafeDivide(sinkRate * float(importance *
                        sourceRate * sourceAngularIntegral), float(sinkRate));
120                 ###
121                 # Kick out zero-valued implicit sensitivitys
122                 ###
123                 if not float(implicitSensitivity):
124                     continue;
125                 ###
126                 # Square away summation indices
127                 ###
128                 key = [];
129                 ###
130                 if not sumOverCells:
131                     key.append(cellNumber);
132                 ###
133                 if not sumOverZas:
134                     key.append(za);
135                 ###
136                 if not sumOverReactions:
137                     key.append(reactionNumber);
138                 ###
139                 key = tuple(key);
140                 ###
141                 try:
142                     key2ImplicitSensitivity[key] += implicitSensitivity;
143                 except KeyError:
144                     key2ImplicitSensitivity[key] = implicitSensitivity;
145         ###
146         # Check implicit sensitivity discrepancies
147         ###
148         discrepancy = sum(ed.GetTotalElement() for ed in key2ImplicitSensitivity.
                values() if ed)
149         if abs(discrepancy) > 1e-3:
150             Warning('{:+.2%} of implicit sensitivity is unaccounted for in response
                    '{}' with single-isotope reaction rates: {}!'.format(discrepancy,
                    response, ', '.join(str(key) for key in sorted(key2ImplicitSensitivity
                    ))));
151         ###
152         return key2ImplicitSensitivity;
```

3.3 Editing MCNP6 for elastic scatter tally tagging

At the time of this writing, the latest available version of MCNP6 is Beta 12 and its most relevant documentation is the MCNPX 2.7.E release memorandum [Pelowitz *et al.*, 2011]. Among the special designations for tally tags described in the document is an 'elastic-scattered particle tag' `0`, designated for particles that have participated in an elastic scatter. Upon inspection of the moniker '0', it is clear that it contains zero entropy bits; it cannot contain enough information to identify a specific cell number, element, or isotope in which the elastic scatter occurs. Other than with this special designator, there is no way to filter particles that have elastically scattered.

A patch composed of seven edits to five files (artf20729) was made to MCNP6, enabling the new feature of the `FTn TAG 4`. Within `colidn.F90` special treatment was written for elastic scatters when `TAG=4`. Within `itally.F90` the input error checking was modified to allow for `FTn TAG` values up to four. Within `annihilation_gammas.F90`, `brems.F90`, and `ttbr.F90`, conditions specific to bremsstrahlung and annihilation photons for `TAG=2,3` were removed.

Using this new tag type, particles undergoing elastic scatter in cell number `CCCCC` with isotope `ZZAAA` are assigned the production tag `CCCCCZZAAA.00002`, instead of `0`. Consequently, elastic scatters are able to join inelastic scatters, $(\mathrm{n}, 2\mathrm{n})$, $(\mathrm{n}, 3\mathrm{n})$, and fissions as possible birth reactions that can be tally tagged.

3.4 Constructing the adjoint source

In order to perform an adjoint transport calculation, the adjoint source must be constructed with the proper distribution, units, and normalization. The adjoint source is defined in Equation 2.37 as:

$$\mathbb{S}^{\dagger} \equiv \frac{\partial R}{\partial \psi}. \tag{3.46}$$

If working with ϕ rather than ψ is desired (scalar flux rather than angular flux), one can use the chain rule:

$$\frac{\partial R}{\partial \psi} = \frac{\partial R}{\partial \phi}\frac{\partial \phi}{\partial \psi}, \tag{3.47}$$

and represent the last partial derivative as a Gâteaux derivative:

$$\frac{\partial \phi}{\partial \psi} = \frac{\frac{d}{dt}\phi(\psi + t\delta\psi)|_{t=0}}{\delta\psi} = \frac{\frac{\partial \phi(\psi)}{\partial \psi}\delta\psi}{\delta\psi}. \tag{3.48}$$

From 3.13 it is clear that ϕ is linear in ψ, so:

$$\frac{\partial \phi}{\partial \psi} = \frac{\phi(\delta\psi)}{\delta\psi}. \tag{3.49}$$

Since $\delta\psi$ can move through the directional integral:

$$\frac{\partial \phi}{\partial \psi} = 4\pi. \tag{3.50}$$

In a subtle way, this means that the adjoint source is isotropic; the presence of neutrons is all that matters for responses, not the direction of travel. If one wishes to convert between an angular adjoint source $\mathbb{S}^\dagger(\vec{r}, E, \vec{\Omega})$ and a scalar adjoint source $\mathbb{S}^\dagger(\vec{r}, E)$, a factor of 4π is all that is needed.

For linear functional responses with the form of Equation 2.16, the angular adjoint source of Equation 2.48 can be converted to a scalar adjoint source, distributed over $(\vec{r}, E)$ in neutron phase space:

$$\mathbb{S}^\dagger = \left(4\pi \frac{R}{\langle \mathbb{H}\psi \rangle}\right) \mathbb{H} = \left(\frac{4\pi R}{\langle \mathbb{H}\psi \rangle}\right) \frac{\mathbb{H}\phi}{\phi}. \tag{3.51}$$

Equivalently, for ratios of linear functional responses with the form of Equation 2.21, the angular adjoint source of Equation 2.50 can be converted to a scalar adjoint source, distributed over $(\vec{r}, E)$ in neutron phase space:

$$\mathbb{S}^\dagger = \left(4\pi \frac{R}{\langle \mathbb{H}_1\psi \rangle}\right) \mathbb{H}_1 - \left(4\pi \frac{R}{\langle \mathbb{H}_2\psi \rangle}\right) \mathbb{H}_2 = \left(\frac{4\pi R}{\langle \mathbb{H}_1\psi \rangle}\right) \frac{\mathbb{H}_1\phi}{\phi} - \left(\frac{4\pi R}{\langle \mathbb{H}_2\psi \rangle}\right) \frac{\mathbb{H}_2\phi}{\phi}. \tag{3.52}$$

Upon inspection of these equations, it is evident that contributions from response dividends are positive and those from response divisors (if they exist) are negative. This presents the likely possibility of locally and/or globally negative adjoint sources. Negative source weights cannot be used in a calculation. However, the importance scales linearly with the adjoint source; the importance can be extracted from a single calculation or summed over any number of independent adjoint calculations given portions of the adjoint source. Knowing this, one calculation is automatically performed for just the dividend and one is performed for just the divisor, if it exists. The results are then superimposed, giving the dividend importance positive sign and the divisor negative sign.

To summarize, the algorithm for extracting adjoint sources for a given response takes its dividend and divisor (if it exists) and iterates over all F^{M4} cell flux multiplier tallies within a forward MCNP6 calculation, finding those that match with the constituents of the response and adjusting for the required multiplicity and sign. These linear functionals are accumulated over all isotopes, but binned over cell and energy. The results are then multiplied by 4π times the total response and divided by the total of that linear functional

times each cell's scalar flux. The `GetCellAdjointSourceEnergyDistribution` method of the `McnpOutputFile` class in `ParseMcnp.py` is responsible for these steps and its source code is included below to allay any confusion:

Algorithm 3.6: GetCellAdjointSourceEnergyDistribution

```
    def GetCellAdjointSourceEnergyDistribution(self,
        cellNumberMaterialNumberReactionNumbers):
        ###
        assert(self.GetIsForward());
        ###
        # Check that all quantitys exist within the same cell
        ###
        assert(1 == len({item[0] for item in cellNumberMaterialNumberReactionNumbers})
            );
        ###
        # Sum of cell reaction rates
        ###
        reactionRate = sum(self.GetCellReactionRateEnergyDistribution(cellNumber,
            materialNumber, reactionNumber) for cellNumber, materialNumber,
            reactionNumber in cellNumberMaterialNumberReactionNumbers);
        reactionRate *= 4 * pi;
        ###
        # Cell track length and volume
        ###
        cellNumber = cellNumberMaterialNumberReactionNumbers[0][0];
        trackLength, volume = self.GetCellTrackLengthVolumeEnergyDistribution(
            cellNumber);
        ###
        totalTrackLength = 0.;
        if isinstance(trackLength, EnergyDistribution):
            totalTrackLength = trackLength.GetTotalElement();
        ###
        totalReactionRate = adjointSource = 0.;
        if isinstance(reactionRate, EnergyDistribution):
            totalReactionRate = reactionRate.GetTotalElement();
            adjointSource = SafeDivide(reactionRate * volume, trackLength);
        ###
        # Interpolate non-zero values below 2e-8
        ###
        adjointSource.ZeroTerp();
        ###
        return (totalReactionRate, totalTrackLength, volume, adjointSource);
```

Within the `WriteAdjointInput` method of the `McnpOutputFile` class in `ParseMcnp.py`, the weights and distributions of adjoint sources are calculated and formatted for adjoint input files. In some instances, the adjoint source must be summed over regions of neutron phase space: leaf cells are summed for their contributions to root cells (cell universes are organized in tree data structures [Weisstein, 2011d]), and energy distributions are summed over energy bins. In both cases, the procedure is to divide the total response by the total neutron track length and multiplying by the total volume, or equivalently, to divide the total

response by the average neutron flux:

$$\mathbb{S}^{\dagger} = \frac{\sum_c \sum_g R_c(E_g)}{\sum_c V_c \sum_g \phi_c(E_g)} \sum_c V_c, \tag{3.53}$$

where c and g are the cell and energy bin indices, respectively. Further inquiry is directed to the method's source below:

Algorithm 3.7: WriteAdjointInput

```
    def WriteAdjointInput(self):
        def CellPath2String(cellPath):
            pathString = '{}';
            if len(cellPath) > 1:
                pathString = '({})';
            ###
            return pathString.format(' < '.join(' '.join(str(cellNumber) for
                cellNumber in cellNumbers) for cellNumbers in cellPath));
        ###
        def CellPath2PathLengthRange(cellPath):
            numberOfPaths = 1;
            for cells in cellPath:
                numberOfPaths *= len(cells);
            return range(numberOfPaths);
        ###
        # Reset automatic input
        ###
        self.ResetAutoInput();
        ###
        # Append automatic input filename
        ###
        self.AppendAutoInputFileName('adj');
        ###
        # cell number --> radii
        ###
        rootCellNumber2Radii = {};
        for rootCell in self.GetCells():
            ###
            # Kick out non-rooted cells
            # The PDS (position direction sampling) coordinate systems are defined
                only for root cells and their radii
            ###
            if rootCell.GetUniverse():
                continue;
            ###
            radii = sorted(surface.GetDimension() for surface in self.FindCellSurfaces
                (rootCell.GetNumber()) if surface.GetIsSphere());
            if 1 == len(radii):
                radii.append(0.);
                radii.reverse();
            ###
            rootCellNumber2Radii[rootCell.GetNumber()] = radii;
        ###
        mnemonic2NewLines = {};
        ###
        # Necessary adjoint tallys
        ###
        necessary = self.FindNecessaryTallys(doAdjoint = True);
```

```
46          ###
47          # F1 and F4 tally lines
48          ###
49          tallyNumber = 0;
50          for mnemonic in ('f1', 'f4'):
51              mnemonic2NewLines[mnemonic] = [WordArrange(words = sorted(necessary[
                    mnemonic]), prefix = 'f{}:n'.format(10 * tallyNumber + int(float(
                    mnemonic[-1]))))];
52          ###
53          # Build missing angle bin lines
54          ###
55          numberOfAngleBins = senseFile.GetParameter('numberOfAngleBins');
56          angleBins = Array(list(-1. + 2. * (index + 1) / numberOfAngleBins for index in
                range(int(numberOfAngleBins))))
57          mnemonic2NewLines['angleBins'] = [WordArrange(words = angleBins, format = '
                {:+.2f}', prefix = 'c0  ')];
58          ###
59          # Build missing energy bin lines
60          ###
61          energyBinsPerDecade = senseFile.GetParameter('energyBinsPerDecade');
62          energyBins = LogFloats(perDecade = int(energyBinsPerDecade));
63          mnemonic2NewLines['energyBins'] = [WordArrange(words = energyBins, format = '
                {:.4E}', prefix = 'e0  ')];
64          ###
65          # Add/replace nps card
66          ###
67          nps = 1e8;
68          npsCard = 'nps {:G}'.format(nps);
69          self.ReplaceAddAutoInputCard('nps', npsCard);
70          ###
71          # Delete kcode and ksrc cards
72          ###
73          self.DeleteAutoInputCard('kcode');
74          self.DeleteAutoInputCard('ksrc');
75          ###
76          # Add/replace mode card
77          ###
78          modeCard = 'mode n';
79          self.ReplaceAddAutoInputCard('mode', modeCard);
80          ###
81          # Add/replace mgopt card
82          ###
83          mgoptCard = 'mgopt a {:G} $ {}'.format(senseFile.GetParameter('
                numberOfEnergyBins') - 1, modeCard);
84          self.ReplaceAddAutoInputCard('mgopt', mgoptCard);
85          ###
86          # Convert library suffixes
87          ###
88          multiGroupSuffix = '.{:3}'.format(LibrarySuffix(isPointWise = False,
                temperature = senseFile.GetParameter('systemTemperature')));
89          self.GetMcnpInputFile().autoInputRaw = ReCompile(r'\.\d\dc', 2 | 8).sub(
                multiGroupSuffix, self.GetAutoInputRaw());
90          ###
91          # Add/replace cut card
92          ###
93          cutCard = 'cut:n j 20.0';
94          self.ReplaceAddAutoInputCard('cut', cutCard);
95          ###
96          # Delete all forward tallies
97          # Delete tally comment cards, thermal scattering libraries, and tally tags
```

```
98          ###
99          deleteCards = [card for card in self.GetTallys()];
100         deleteCards.extend(card for cardName in ('tallyComments', 'thermalScatters', '
                energys', 'angles') for card in self.GetNamedCard(cardName));
101         deleteCards.extend(card for tallyTag in self.GetNamedCard('tallyTags').values
                () for card in tallyTag);
102         for card in deleteCards:
103             self.GetMcnpInputFile().autoInputRaw = card.GetRegex().sub('', self.
                    GetAutoInputRaw());
104         ###
105         # Append new lines to auto input raw
106         ###
107         if mnemonic2NewLines:
108             self.ReplaceAddAutoInputCard(None, '\n'.join('\n'.join(newLines) for
                    mnemonic, newLines in sorted(mnemonic2NewLines.items())));
109         ###
110         # Iterate over responses
111         ###
112         autoInputRaw = self.GetAutoInputRaw();
113         for response in senseFile.GetResponses():
114             ###
115             # Extract adjoint source distributions
116             ###
117             cellNumberSign2ReactionRateTrackLengthVolumeAdjointSource = self.
                    GetResponseAdjointSources(response, includeAll = True);
118             ###
119             # Adjoint input for divisor and dividend
120             ###
121             for sign in (+1, -1):
122                 ###
123                 # Reset automatic input
124                 ###
125                 self.GetMcnpInputFile().autoInputRaw = autoInputRaw;
126                 ###
127                 # Filter adjoint sources according to sign
128                 ###
129                 cellNumber2ReactionRateTrackLengthVolumeAdjointSource = {cellNumber :
                        value for (cellNumber, tempSign), value in
                        cellNumberSign2ReactionRateTrackLengthVolumeAdjointSource.items()
                        if tempSign == sign};
130                 ###
131                 # Kick out if no cells have adjoint sources
132                 # This can happen if the necessary tallies are missing, or if reaction
                        rates are zero-valued
133                 ###
134                 if not cellNumber2ReactionRateTrackLengthVolumeAdjointSource:
135                     continue;
136                 ###
137                 # Generate cell mappings
138                 ###
139                 cellNumber2ErgDistributionNumber = {};
140                 cellNumber2RadDistributionNumber = {};
141                 ###
142                 cellNumber2AdjointSource = {};
143                 cellNumber2TotalAdjointSource = {};
144                 cellNumber2Rtv = {};
145                 cellNumber2Paths = {};
146                 ###
147                 distributionNumber = max((1, 2, 3)) + 1;
148                 for cellNumber, (reactionRate, trackLength, volume, totalAdjointSource
```

```
                    , adjointSource) in sorted(
                    cellNumber2ReactionRateTrackLengthVolumeAdjointSource.items()):
149                 ###
150                 # Each leaf cell has its own energy distribution
151                 ###
152                 if cellNumber not in cellNumber2ErgDistributionNumber:
153                     cellNumber2ErgDistributionNumber[cellNumber] =
                            distributionNumber;
154                     ###
155                     # Increment to unique number
156                     ###
157                     distributionNumber += 1;
158                 ###
159                 # Each root cell has its own radii
160                 ###
161                 for rootCell in self.FindRootCells(cellNumber):
162                     if rootCell.GetNumber() not in
                            cellNumber2RadDistributionNumber:
163                         cellNumber2RadDistributionNumber[rootCell.GetNumber()] =
                                distributionNumber;
164                         ###
165                         # Increment to unique number
166                         ###
167                         distributionNumber += 1;
168                 ###
169                 # cell number --> adjoint source energy distribution
170                 ###
171                 cellNumber2AdjointSource[cellNumber] = adjointSource
172                 ###
173                 # cell number --> total adjoint source
174                 ###
175                 cellNumber2TotalAdjointSource[cellNumber] = totalAdjointSource;
176                 ###
177                 # cell number --> (reaction rate, track length, volume)
178                 ###
179                 cellNumber2Rtv[cellNumber] = (reactionRate, trackLength, volume)
180                 ###
181                 # cell number --> cell paths
182                 ###
183                 cellNumber2Paths[cellNumber] = self.GetCellNumberPaths(cellNumber)
                        ;
184             ###
185             # System adjoint source
186             ###
187             systemReactionRate = 0.;
188             systemTrackLength = 0.;
189             systemVolume = 0.;
190             for reactionRate, trackLength, volume in cellNumber2Rtv.values():
191                 systemReactionRate += reactionRate;
192                 systemTrackLength += trackLength;
193                 systemVolume += volume;
194             ###
195             cellNumberPaths = sorted((cellNumber, cellPaths) for cellNumber,
                    cellPaths in sorted(cellNumber2Paths.items()));
196             ###
197             # sdef card
198             ###
199             sdefCard = '';
200             ###
201             # sdef line
```

```
202                 ###
203                 sdefCard += 'sdef cel d{} rad fcel d{} erg fcel d{} wgt {:G}'.format
                        (1, 2, 3, 1.);
204                 ###
205                 # cel path's
206                 ###
207                 sdefCard += WordArrange(words = (CellPath2String(cellPath) for
                        cellNumber, cellPaths in cellNumberPaths for cellPath in cellPaths
                        ), prefix = '\nsi{} l'.format(1));
208                 ###
209                 # leaf cell wgt's
210                 ###
211                 sdefCard += WordArrange(words = (cellNumber2TotalAdjointSource[
                        cellNumber] for cellNumber, cellPaths in cellNumberPaths for
                        cellPath in cellPaths for pathNumber in CellPath2PathLengthRange(
                        cellPath)), format = '{:G}', prefix = '\nsp{} d'.format(1));
212                 ###
213                 # leaf cell root cell rad distribution indices
214                 ###
215                 sdefCard += WordArrange(words = (cellNumber2RadDistributionNumber[
                        rootCellNumber] for cellNumber, cellPaths in cellNumberPaths for
                        cellPath in cellPaths for pathNumber in CellPath2PathLengthRange(
                        cellPath) for rootCellNumber in (abs(cellPath[-1][-1]), )), prefix
                         = '\nds{} s'.format(2));
216                 ###
217                 # leaf cell root cell rad distributions
218                 ###
219                 rootCellNumbers = sorted({rootCellNumber for cellNumber, cellPaths in
                        cellNumberPaths for cellPath in cellPaths for rootCellNumber in (
                        abs(cellPath[-1][-1]), )});
220                 for rootCellNumber in rootCellNumbers:
221                     sdefCard += WordArrange(words = rootCellNumber2Radii[
                            rootCellNumber], format = '{:G}', prefix = '\nsi{}'.format(
                            cellNumber2RadDistributionNumber[rootCellNumber]));
222                 ###
223                 # leaf cell erg distribution indices
224                 ###
225                 sdefCard += WordArrange(words = (cellNumber2ErgDistributionNumber[
                        cellNumber] for cellNumber, cellPaths in cellNumberPaths for
                        cellPath in cellPaths for pathNumber in CellPath2PathLengthRange(
                        cellPath)), prefix = '\nds{} s'.format(3));
226                 ###
227                 # leaf cell erg distributions
228                 ###
229                 for cellNumber, cellPaths in cellNumberPaths:
230                     sdefCard += WordArrange(words = cellNumber2AdjointSource[
                            cellNumber].GetEnergys(), format = '{:.4E}', prefix = '\nsi{}
                            h 0.0'.format(cellNumber2ErgDistributionNumber[cellNumber]));
231                     sdefCard += WordArrange(words = cellNumber2AdjointSource[
                            cellNumber].GetElements(), format = '{:.4E}', prefix = '\nsp{}
                            d 0.0'.format(cellNumber2ErgDistributionNumber[cellNumber]));
232                 ###
233                 self.ReplaceAddAutoInputCard('sdef', sdefCard);
234                 self.ReplaceAddAutoInputCard('nps', str(sign));
235                 ###
236                 # Adjoint input filename
237                 ###
238                 fileName = response.GetFileName(self.GetAutoInputFileName(), sign);
239                 ###
240                 # Write adjoint input file
```

```
241                 ###
242                 WriteFile(fileName, self.GetAutoInputRaw());
243         return;
```

3.5 Extracting the importance

Having performed an adjoint transport calculation with the proper source, the importance needs to be extracted with the correct units. A dimensional analysis of the terms in equations 2.3 and 2.37 is sufficient to determine the units of an F^4 cell flux tally within an adjoint transport calculation.

The units of $\mathbb{A}$, $\mathbb{S}$, and ψ within Equation 2.3 are:

$$[\mathbb{A}] = [\Sigma] = \left[\frac{\text{birth} - \text{neutron}}{\text{cm} \cdot \text{neutron}}\right], \tag{3.54}$$

$$[\mathbb{S}] = \left[\frac{\text{birth} - \text{neutron}}{\text{cm}^3 \cdot \text{source} \cdot \text{MeV} \cdot \text{ster}}\right], \tag{3.55}$$

$$[\psi] = \frac{[\mathbb{S}]}{[\mathbb{A}]} = \left[\frac{\text{birth} - \text{neutron}}{\text{cm}^3 \cdot \text{source} \cdot \text{MeV} \cdot \text{ster}} \cdot \frac{\text{cm} \cdot \text{neutron}}{\text{birth} - \text{neutron}}\right] = \left[\frac{\text{neutron}}{\text{cm}^2 \cdot \text{source} \cdot \text{MeV} \cdot \text{ster}}\right]. \tag{3.56}$$

F^4 tallies extract cell scalar fluxes and have the units of ψ multiplied by 4π [X-5 Monte Carlo Team, 2005, p. 2-81], which can be generalized as:

$$\left[\mathrm{F}^4\right] = \frac{[\mathbb{S}]}{[\mathbb{A}]} \times [4\pi]. \tag{3.57}$$

Since $\mathbb{A}^\dagger$ is the adjoint of $\mathbb{A}$, it has the same units:

$$\left[\mathbb{A}^\dagger\right] = [\mathbb{A}]. \tag{3.58}$$

From Equation 2.37:

$$\left[\mathbb{S}^\dagger\right] = \frac{[R]}{[\psi]} = \left[\frac{\text{response} \cdot \text{cm}^2 \cdot \text{ebin} \cdot \text{ster}}{\text{neutron}}\right]. \tag{3.59}$$

However, when $\mathbb{S}^\dagger$ is a volumetric source distributed over a V, over E bins, and over $\vec{\Omega}$ bins:

$$\left[\mathbb{S}^\dagger\right] = \left[\frac{\text{response} \cdot \text{cm}^2 \cdot \text{ebin} \cdot \text{ster}}{\text{neutron}} \cdot \frac{1}{\text{cm}^3 \cdot \text{ebin} \cdot \text{ster}}\right] = \left[\frac{\text{response}}{\text{cm} \cdot \text{neutron}}\right]. \tag{3.60}$$

Using the generalization of F^4 tally units from Equation 3.57 for adjoint transport calculations:

$$\begin{aligned}\left[\mathrm{F}^{4\dagger}\right] = \frac{\left[\mathbb{S}^\dagger\right]}{\left[\mathbb{A}^\dagger\right]} \times [4\pi] = \left[\frac{\text{response}}{\text{cm}\cdot\text{neutron}} \cdot \frac{\text{cm}\cdot\text{neutron}}{\text{birth}-\text{neutron}} \cdot \text{ster}\right] \\ = \left[\frac{\text{response}\cdot\text{ster}}{\text{birth}-\text{neutron}}\right].\end{aligned} \tag{3.61}$$

In effect, $F^{4\dagger}$ cell scalar flux tallies for adjoint transport calculations with adjoint sources quantify the importance times steradians:

$$\phi_c^\dagger(E) = \frac{F_c^{4\dagger}}{4\pi}. \tag{3.62}$$

When collapsing importances over a region of neutron phase space, weighted averages must always be taken. For example, combining the importance of multiple cells, the importances are multiplied by their respective volumes, summed, and then divided by the total volume. Energy and directional bin widths are treated in the same way.

One last normalization is performed for importances. Using the relations:

$$\left\langle \psi^\dagger, \mathbb{S} \right\rangle = \left\langle \psi^\dagger, \mathbb{A}\psi \right\rangle = \left\langle \mathbb{A}^\dagger\psi^\dagger, \psi \right\rangle = \left\langle \mathbb{S}^\dagger, \psi \right\rangle = \left\langle \mathbb{H}\psi \right\rangle = R, \tag{3.63}$$

responses in the denominators of Equations 2.47 and 2.49 can be replaced with $\left\langle \psi^\dagger, \mathbb{S} \right\rangle$. This allows any incorrect magnitudes or factors in the importance (such as 4π) to cancel out. For example, when $\mathbb{S}$ is a fusion point source:

$$\mathbb{S} = \frac{\delta(E-E_0)\delta(\vec{r}-\vec{r}_0)}{4\pi}, \tag{3.64}$$

and $\left\langle \psi^\dagger, \mathbb{S} \right\rangle$ becomes $\frac{1}{4\pi}\psi^\dagger(\vec{r}=\vec{r}_0, E=E_0)$. All importances are automatically normalized by this factor in the `McnpOutputFile` class method `GetCellImportancePerResponseEnergyDistribution`, shown below:

Algorithm 3.8: GetCellImportancePerResponseEnergyDistribution

```
1    def GetCellImportancePerResponseEnergyDistribution(self, cellNumber, particles='
         np'):
2        ###
3        assert(not self.GetIsForward());
4        ###
5        # Extract adjoint flux (importance)
6        ###
7        importance = self.GetCellScalarFluxEnergyDistribution(cellNumber, particles);
8        ###
9        # Normalize importance
10       ###
11       fusionChamberImportance = self.GetCellScalarFluxEnergyDistribution(senseFile.
```

```
            GetParameter('fusionChamberCellNumber'), particles);
12          divisor = fusionChamberImportance.LinTerp(senseFile.GetParameter('
                fusionNeutronEnergy'));
13          ###
14          return importance * SafeDivide(1., divisor * 4. * pi);
```

3.6 Figure of merit extraction loose ends

Examples of linear functional responses are the tritium breeding ratio, displacements per atom of a structural material, gaseous production rates, thermal heating rates, and the activation rate of a problematic radioisotope. Ignoring the uncertainty in constants such as Q-values and source rates, these quantities (and their uncertainties) can all be extracted from MCNP6 in exactly the manner that is described in previous sections. This section addresses the somewhat more complicated linear function ratio responses: the fissile fuel conversion ratio, blanket multiplication, and k_{eff}.

The fissile fuel conversion ratio is generally expressed as:

$$\mathrm{CR} \equiv \frac{\text{fertile} \rightarrow \text{fissile rate}}{\text{fissile destruction rate}}. \tag{3.65}$$

While fissile isotopes can be defined as either "isotopes able to sustain nuclear chain reactions" or "isotopes whose fission cross-sections have no threshold," the identification of fertile isotopes is sometimes contested. Because of this, a simple algorithm was devised to find all fertile parents of a given fissile isotope, with reasonable assumptions. First, it is assumed that a fertile breeding chain can originate only from a single radiative capture upon a long-lived isotope. Second, the chain, which terminates at a fissile isotope, must consist only of short-lived isotopes. Capture and decay relational mappings are contained in `fertileDecayFroms`. These mappings are traversed upwards in a 'fertile quick-decay tree' starting at a fissile isotope (`fissileZas`) in a depth-first fashion with the function `Za2FertileParents`, accumulating fertile parents that reside at the roots of the fertile breeding tree. The source code for the list, mapping, and function are shown below:

Algorithm 3.9: fissileZas

```
1  ###
2  # List of fissile za's
3  ###
4  fissileZas = (\
5      90227,\
6      90229,\
7      92233,\
8      92235,\
9      93237,\
10     94239,\
11     94241,\
12     95241,\
```

```
13      96243,\
14      96245,\
15      96247,\
16      98249,\
17      );
```

Algorithm 3.10: fertileDecayFroms

```
1  ###
2  # Members of fertile quick-decay trees that originate from a single (n, \gamma)
       reaction upon a long-lived isotope
3  ###
4  fertileDecayFroms = {\
5       90228 : 89228,\
6       91231 : 90231,\
7       91233 : 90233,\
8       92232 : 91232,\
9       92233 : 91233,\
10      93237 : 92237,\
11      93239 : 92239,\
12      94238 : 93238,\
13      94239 : 93239,\
14      95243 : 94243,\
15      95245 : 94245,\
16      96244 : 95244,\
17      96245 : 95245,\
18      97249 : 96249,\
19      97251 : 96251,\
20      98249 : 97249,\
21      98250 : 97250,\
22      98251 : 97251,\
23      };
```

Algorithm 3.11: Za2FertileParents

```
1  ###
2  # Follow the fertile quick-decay tree to an originating isotope
3  ###
4  def Za2FertileParents(za):
5       output = za;
6       while output in fertileDecayFroms:
7            output = fertileDecayFroms[output];
8       output = [output];
9       if za not in output:
10           output.append(za);
11      return [za - 1 for za in output];
```

Blanket neutron multiplication concerns the neutron economy of a blanket exposed to an external source. It is usually defined as:

$$\mathrm{M}_n \equiv \frac{\mathrm{S}+\mathrm{P}}{\mathrm{S}} = 1 + \frac{\mathrm{P}}{\mathrm{S}}, \tag{3.66}$$

where S and P are the net neutron source and neutron production rates, respectively. Because all production rates are tallied on a per source particle basis, the blanket neutron multiplication becomes the sum of all source reactions multiplied by their respective multiplicities,

plus one.

$$\mathrm{M}_n = 1 + \sum_c \sum_z \sum_x (\nu_{z,x} - 1)\ F^{M4}_{c,z,x}. \tag{3.67}$$

If the blanket thermal multiplication is desired, the multiplicity can be replaced by the Q-value for the specific reaction divided by the energy of source neutrons:

$$\mathrm{M}_{th} = 1 + \sum_c \sum_z \sum_x \frac{Q_{z,x}}{E_{source}}\ F^{M4}_{c,z,x}. \tag{3.68}$$

k_{eff} is a different measure of neutron economy, quantifying the number of neutrons that are produced per neutron absorbed:

$$k_{eff} \equiv \frac{\mathrm{P}}{\mathrm{A}}, \tag{3.69}$$

where A is the neutron absorption rate. The fate of all neutrons is either to leak or to be absorbed, so:

$$k_{eff} = \frac{\mathrm{P}}{\mathrm{S} - \mathrm{L}} = \frac{\mathrm{P/S}}{1 - \frac{\mathrm{L}}{\mathrm{S}}} = \frac{\mathrm{M} - 1}{1 - \frac{\mathrm{L}}{\mathrm{S}}}. \tag{3.70}$$

Leakage rates are also tallied on a per source particle basis, so k_{eff} becomes:

$$k_{eff} = \frac{\mathrm{M} - 1}{1 - \sum_s \sum_{\mu>0} F^1_{s,\mu}}, \tag{3.71}$$

where s is the index of surfaces at the outer extremity of the system and μ is the surface directional cosine bin. It should be noted that this interpretation of k_{eff} is different from the traditional one [Cullen, 2009; Kiedrowski and Brown, 2010]. Source neutrons start from the external source instead of an assumed fission spectrum that is artificially depressed by a constant. If the latter interpretation of k_{eff} is desired, an MCNP6 `kcode` calculation can instead be performed.

Not all isotopes have fission or tritium-production cross-sections. In order to avoid the needless tallying of zero-valued reactions, a function `IsZaReactionNumberOfInterest` was written (inspired by Monteburns [Poston and Trellue, 1999]) that is based upon the nuclear data that exists within `ENDF-B/VII.0`. The function returns whether a given isotope and nuclear reaction type can be of interest–if the cross-sections exist and don't have an extremely high threshold energy. Independent metrics, such as the abundance of the isotope can additionally determine whether they can be of interest. The algorithm is shown below:

Algorithm 3.12: IsZaReactionNumberOfInterest

```
1 ###
2 # Reaction numbers of interest and the za's that cause them
3 # Based upon ENDF/B-VII.0, Low-fidelity covariance library
```

```
4  ###
5  def IsZaReactionNumberOfInterest(za, reactionNumber):
6      z, a = za // 1000, za % 1000;
7      ###
8      # (n, n) can be significant for any isotope
9      ###
10     if 2 == reactionNumber:
11         return True;
12     ###
13     # (n, n') can be significant for any isotope, but does not exist for some
14     ###
15     if 4 == reactionNumber:
16         return za not in (1001, 1002, 1003, 2003, 2004, 4007, 4009, 23000, 28059,
               33074, 39090, 91231, 91233, 98253, 99253);
17     ###
18     # (n, 2n) can be significant for any isotope, but does not exist for some
19     ###
20     if 16 == reactionNumber:
21         return za not in (1001, 2003, 2004, 3006, 4007, 5010, 6000, 98253, 99253);
22     ###
23     # (n, 3n) is assumed only significant for actinides, but does not exist for some
24     ###
25     if 17 == reactionNumber:
26         return z > 88 and za not in (92234, 92236, 93237, 94238, 94240, 94241, 94242,
               95241, 95242, 95243, 96242, 96243, 96244, 96245, 98253, 99253);
27     ###
28     # (n, fission) is assumed only significant for actinides, but does not exist for
           some
29     ###
30     if 18 == reactionNumber:
31         return z > 88 and za not in (89225, 89226, 99253);
32     ###
33     # (n, \gamma) can be significant for any isotope, but does not exist for some
34     ###
35     if 102 == reactionNumber:
36         return za not in (1003, 2004, 4007);
37     ###
38     # (n, p) is assumed only significant for fission products, but does not exist for
           some
39     ###
40     if 103 == reactionNumber:
41         return z < 89 and za not in (1001, 1002, 1003, 2004, 3007, 4007);
42     ###
43     # (n, t) is significant below 14.1 MeV for these isotopes
44     ###
45     if 205 == reactionNumber:
46         return za in (3006, 3007, 4009, 5011, 7014, 7015, 9019);
47     ###
48     # MCNP's -6 is equivalent to MT = 18
49     ###
50     if -6 == reactionNumber:
51         return IsZaReactionNumberOfInterest(za, 18);
52     ###
53     # 205 is a better MT than 105
54     ###
55     if 105 == reactionNumber:
56         return IsZaReactionNumberOfInterest(za, 205);
57     ###
58     # Unaccounted-for reactions are assumed to be not of interest
59     ###
```

```
60      return False;
```

3.7 Accounting for Monte Carlo counting uncertainty

One universal property of Monte Carlo simulations is that their results are subject to counting statistics; there is an inherent aleatory uncertainty that depends upon the number of samples that are performed [Knoll, 2000, p. 83]. As in most realms of statistics, the true counting precision cannot be known and must be estimated by employing a statistical model that is an analogue to the simulation. If Poisson or Gaussian models are used, the counting variance is estimated as:

$$\sigma^2 = N, \tag{3.72}$$

and the relative standard error is:

$$\frac{\sigma}{N} = \frac{1}{\sqrt{N}}, \tag{3.73}$$

where N is the number of counted successes (particle hits). The most reliable way to decrease counting uncertainty is to increase the number of sample histories.

MCNP6 estimates the counting uncertainty of all results and reports them alongside the expected values [X-5 Monte Carlo Team, 2005, p. 1-6,2-108]. These results are inevitably used to calculate derived quantities by way of summations over energy bins or cells, ratios, or products. The uncertainty that the derived quantities inherit from underlying terms must be propagated regardless of how they are derived.

3.7.1 Counting uncertainty propagation

Because virtually all results that `ParseMcnp.py` extracts from MCNP6 are energy distributions with counting uncertainties, an uncertain energy distribution class `EnergyDistribution` was written. All binary arithmetic operators in the class are written so that the uncertainty (along with expected values) from both objects involved in an operation is propagated correctly to the result which is also an `EnergyDistribution` object. The standard rules are used for error propagation with random variables [Weisstein, 2011b]. When $u \equiv x \pm c$:

$$\sigma_u^2 = \sigma_x^2. \tag{3.74}$$

When $u \equiv x \pm y$:

$$\sigma_u^2 = \sigma_x^2 + \sigma_y^2 + 2\sigma_x\sigma_y\rho_{xy}, \tag{3.75}$$

where ρ_{xy} is the correlation coefficient between x and y. When $u \equiv xc$:

$$\sigma_u^2 = c^2\sigma_x^2. \tag{3.76}$$

When $u \equiv xy$:

$$\left(\frac{\sigma_u}{u}\right)^2 = \left(\frac{\sigma_x}{x}\right)^2 + \left(\frac{\sigma_y}{y}\right)^2 + \frac{2\sigma_x\sigma_y}{xy}\rho_{xy}. \tag{3.77}$$

When $u \equiv x/c$:

$$\sigma_u^2 = \frac{1}{c^2}\sigma_x^2. \tag{3.78}$$

When $u \equiv x/y$:

$$\left(\frac{\sigma_u}{u}\right)^2 = \left(\frac{\sigma_x}{x}\right)^2 + \left(\frac{\sigma_y}{y}\right)^2 - \frac{2\sigma_x\sigma_y}{xy}\rho_{xy}. \tag{3.79}$$

Other methods are also attached to the `EnergyDistribution` class, such as getters for distributions per unit energy and lethargy, down-sampling methods, and interpolation methods. Further interest is directed towards the source below:

Algorithm 3.13: EnergyDistribution

```
###
# Uncertain energy distribution class
###
class EnergyDistribution:
    def __float__(self):
        return self.GetTotalElement();
    ###
    def __init__(self, *args):
        if 2 == len(args):
            ###
            # Match iterator and number of iterations
            ###
            matches, numberOfIterations = args;
            ###
            index = 0;
            self.energys, self.elements, self.variances = (Empty(numberOfIterations)
                for index in range(3));
            for match in matches:
                try:
                    ###
                    # Energys, elements, variances
                    ###
                    energy, element, relativeUncertainty = (float(number) for number
                        in match.groups());
                    ###
                    self.energys[index], self.elements[index], self.variances[index] =
                        energy, element, element ** 2. * relativeUncertainty ** 2.;
                except ValueError:
                    ###
                    # Total element, total variance
                    ###
                    totalElement, totalRelativeUncertainty = (float(number) for number
                        in match.group(2, 3));
                    ###
                    self.totalElement, self.totalVariance = totalElement, totalElement
                        ** 2. * totalRelativeUncertainty ** 2.;
                ###
                index += 1;
```

```
34          ###
35          elif 5 == len(args):
36              ###
37              # Energys, elements, variances, total element, total variance
38              ###
39              self.energys, self.elements, self.variances, self.totalElement, self.
                    totalVariance = args;
40          return;
41      ###
42      def __len__(self):
43          return len(self.GetEnergys());
44      ###
45      def __str__(self):
46          return '\n'.join('{} {} {}'.format(self.GetEnergys()[index], self.GetElements
                ()[index], self.GetVariances()[index]) for index in range(len(self)));
47      ###
48      # Mathematical operator overloaders
49      ###
50      def __add__(self, other):
51          if isinstance(other, int) or isinstance(other, float):
52              ###
53              # Elements and sum(elements) are affected
54              ###
55              elements = self.GetElements() + other;
56              totalElement = self.GetTotalElement() + other;
57              ###
58              # Variances, and sum(variances) are not affected
59              ###
60              variances = self.GetVariances();
61              totalVariance = self.GetTotalVariance();
62          ###
63          elif isinstance(other, self.__class__):
64              assert(len(self) == len(other) and all(self.GetEnergys() == other.
                    GetEnergys()));
65              ###
66              # var = var_a + var_b + (2 * corr_ab * std_a * std_b)
67              # Correlation is assumed as 0 for addition
68              ###
69              correlationCoefficient = 0.;
70              ###
71              elements = self.GetElements() + other.GetElements();
72              variances = self.GetVariances() + other.GetVariances() + (2. * self.
                    GetUncertaintys() * other.GetUncertaintys() * correlationCoefficient);
73              totalElement = self.GetTotalElement() + other.GetTotalElement();
74              totalVariance = self.GetTotalVariance() + other.GetTotalVariance();
75          ###
76          return EnergyDistribution(self.GetEnergys(), elements, variances, totalElement
                , totalVariance);
77      ###
78      def __mul__(self, other):
79          if isinstance(other, int) or isinstance(other, float):
80              ###
81              # Elements, variances, sum(elements), sum(variances) are affected
82              ###
83              elements = self.GetElements() * other;
84              variances = self.GetVariances() * (other ** 2.);
85              totalElement = self.GetTotalElement() * other;
86              totalVariance = self.GetTotalVariance() * (other ** 2.);
87          elif isinstance(other, self.__class__):
88              assert(len(self) == len(other) and all(self.GetEnergys() == other.
```

```
                GetEnergys()));
89              ###
90              # relvar = relvar_a + relvar_b + (2 * relstd_a * relstd_b * corr_ab)
91              # Correlation is assumed as 1 for multiplication
92              ###
93              correlationCoefficient = +1.;
94              ###
95              elements = self.GetElements() * other.GetElements();
96              variances = elements ** 2. * (self.GetRelativeVariances() + other.
                    GetRelativeVariances() + (2. * self.GetRelativeUncertaintys() * other.
                    GetRelativeUncertaintys() * correlationCoefficient))
97              ###
98              totalElement = sum(elements);
99              ###
100             totalVariance = sum(variances);
101         ###
102         return EnergyDistribution(self.GetEnergys(), elements, variances, totalElement
                , totalVariance);
103     ###
104     def __truediv__(self, other):
105         if isinstance(other, int) or isinstance(other, float):
106             return self * SafeDivide(1., other);
107         elif isinstance(other, self.__class__):
108             assert(len(self) == len(other) and all(self.GetEnergys() == other.
                    GetEnergys()));
109             ###
110             # relvar = relvar_a + relvar_b - (2 * relstd_a * relstd_b * corr_ab)
111             # Correlation is assumed as 1 for division
112             ###
113             correlationCoefficient = +1.;
114             ###
115             elements = self.GetElements() / other.GetElements();
116             variances = elements ** 2. * abs(self.GetRelativeVariances() + other.
                    GetRelativeVariances() - (2. * self.GetRelativeUncertaintys() * other.
                    GetRelativeUncertaintys() * correlationCoefficient));
117             ###
118             # Change nan's to zero's
119             ###
120             elements = Nan2Num(elements);
121             elements[abs(elements) > 1e100] = 0.;
122             variances = Nan2Num(variances);
123             variances[abs(variances) > 1e100] = 0.;
124             ###
125             totalElement = sum(elements);
126             ###
127             totalVariance = sum(variances);
128         ###
129         return EnergyDistribution(self.GetEnergys(), elements, variances, totalElement
                , totalVariance);
130     ###
131     __radd__ = __add__;
132     ###
133     __rmul__ = __mul__;
134     ###
135     def __sub__(self, other):
136         return self.__add__(-other);
137     ###
138     def __rsub__(self, other):
139         return -self.__sub__(other);
140     ###
```

```
141     def __neg__(self):
142         return -1. * self;
143     ###
144     # Generic getter methods
145     ###
146     def GetElements(self):
147         return self.elements;
148     ###
149     def GetEnergys(self):
150         return self.energys;
151     ###
152     def GetTotalElement(self):
153         return self.totalElement;
154     ###
155     def GetTotalVariance(self):
156         return self.totalVariance;
157     ###
158     def GetVariances(self):
159         return self.variances;
160     ###
161     # Algorithmic methods
162     ###
163     def HalfSample(self, doAverage = False):
164         isOdd = len(self) % 2;
165         ###
166         energys = self.GetEnergys()[1 - isOdd : : 2];
167         ###
168         elements = self.GetElements()[1 - isOdd : : 2] + Concatenate(((0., ) * isOdd,
                self.GetElements()[isOdd : : 2]));
169         variances = self.GetVariances()[1 - isOdd : : 2] + Concatenate(((0., ) * isOdd
                , self.GetVariances()[isOdd : : 2]));
170         totalElement = self.GetTotalElement();
171         totalVariance = self.GetTotalVariance();
172         ###
173         if doAverage:
174             elements[isOdd : ] /= 2.;
175             variances[isOdd : ] /= 4.;
176             totalElement /= 2;
177             totalVariance /= 4;
178         ###
179         return EnergyDistribution(energys, elements, variances, totalElement,
                totalVariance);
180     ###
181     def LinTerp(self, energy):
182         energys, means, elements = self.GetEnergys(), self.GetEnergyBinMeans(), self.
                GetElements();
183         ###
184         # Extract non-zero-valued elements 2 decades surrounding energy
185         ###
186         index = LogicalAnd(LogicalAnd(energy / 10 <= energys, energys <= energy * 10),
                 elements != 0);
187         ###
188         # Kick out null non-zeros
189         ###
190         if not len(index):
191             return;
192         ###
193         return LogLinTerp(means[index], elements[index], energy);
194     ###
195     def Resample(self, energys):
```

```
196             indices = [];
197             for energy in energys:
198                 try:
199                     index = Where(self.GetEnergys() >= energy)[0][0];
200                 except IndexError:
201                     index = -1;
202                 ###
203                 indices.append(index);
204             ###
205             lethargyPerBins = -NaturalLogarithm(1. - Concatenate((energys[ : 1],
                    Difference(energys))) / energys);
206             ###
207             elements = lethargyPerBins * self.GetPerLethargys()[indices];
208             variances = lethargyPerBins * SafeDivide(self.GetVariances()[indices], self.
                    GetLethargyPerBins()[indices])
209             ###
210             elements[Where(abs(elements) > 1e100)] = 0.;
211             variances[Where(abs(variances) > 1e100)] = 0.;
212             ###
213             return EnergyDistribution(energys, Nan2Num(elements), Nan2Num(variances), self
                    .GetTotalElement(), self.GetTotalVariance());
214         ###
215         def ZeroTerp(self, limit = 2e-8):
216             energys, elements, variances = self.GetEnergys(), self.GetElements(), self.
                    GetVariances();
217             ###
218             # Extract non-zero-valued and zero-valued elements
219             ###
220             nonZeros = LogicalAnd(energys <= limit, elements != 0);
221             zeros = LogicalAnd(energys <= limit, elements == 0);
222             ###
223             # Kick out null non-zeros
224             ###
225             if not sum(nonZeros):
226                 return;
227             ###
228             # Populate zero-valued elements based upon linear interpolation of the
                    logarithms
229             ###
230             self.elements[zeros] = LogLinTerp(energys[nonZeros], elements[nonZeros],
                    energys[zeros]);
231             ###
232             # Populate zero-valued variances based upon linear interpolation of the
                    logarithms
233             ###
234             self.variances[zeros] = self.elements[zeros] ** 2.;
235             ###
236             return;
237         ###
238         # Derived statistical getter methods
239         ###
240         def GetUncertaintys(self):
241             return self.GetVariances() ** 0.5;
242         ###
243         def GetRelativeUncertaintys(self):
244             return self.GetRelativeVariances() ** 0.5;
245         ###
246         def GetRelativeVariances(self):
247             return Nan2Num(SafeDivide(self.GetVariances(), self.GetElements() ** 2.));
248         ###
```

```
249     def GetTotalRelativeUncertainty(self):
250         return SafeDivide(self.GetTotalVariance(), self.GetTotalElement() ** 2.) **
                0.5;
251     ###
252     # Bin parameter getter methods
253     ###
254     def GetEnergyBinMeans(self):
255         return self.GetEnergys() - 0.5 * self.GetEnergyPerBins();
256     ###
257     def GetEnergyPerBins(self):
258         return Concatenate((self.GetEnergys()[ : 1], Difference(self.GetEnergys())));
259     ###
260     def GetLethargyPerBins(self):
261         return Nan2Num(-NaturalLogarithm(1. - self.GetEnergyPerBins() / self.
                GetEnergys()));
262     ###
263     # Element per getter methods
264     ###
265     GetPerBins = GetElements;
266     ###
267     def GetPerEnergys(self):
268         return SafeDivide(self.GetPerBins(), self.GetEnergyPerBins());
269     ###
270     def GetPerLethargys(self):
271         return SafeDivide(self.GetPerBins(), self.GetLethargyPerBins());
272     ###
273     def GetNormPerLethargys(self):
274         return SafeDivide(self.GetPerLethargys(), self.GetTotalElement());
```

3.7.2 Variance of variance

One quantity that is derived from uncertain physical quantities is the sensitivity of a response to uncertain nuclear data, which is used to calculate the nuclear data relative variance of the response. The estimate of nuclear data uncertainty has its own counting uncertainty; there is counting variance in the nuclear data variance and the two compound each other. In order for an uncertainty analysis to be meaningful, it is necessary to quantify the portion of uncertainty that is from the dispersion in nuclear cross-sections and the portion that is due to Monte Carlo counting uncertainty.

The form of the variance of the variance that is most accessible is the counting variance of the nuclear data relative variance of a quantitiy:

$$z \equiv \mathrm{var}\left[\left(\frac{\mathrm{std}\left[r\right]}{\exp\left[r\right]}\right)^2\right] \equiv \mathrm{var}\left[y\right], \tag{3.80}$$

where z is the 'variance of the variance', y is the expected nuclear data relative variance, and r is the expected value of the quantity. In order to accumulate these variances of variances over several nuclear data uncertainties (e.g. over multiple energy bins or isotopes), the

counting variance of the nuclear data standard deviation is actually required:

$$\mathrm{var}\left[\mathrm{std}\left[r\right]\right] = \mathrm{var}\left[\exp\left[r\right]\ \sqrt{y}\right] = \exp\left[r\right]^2\ \mathrm{var}\left[\sqrt{y}\right]. \tag{3.81}$$

Using a second-order Taylor expansion of the second moment of a function of a random variable [Wikipedia, 2011; Powers, 2010]:

$$\mathrm{var}\left[f\left(x\right)\right] \cong \left(\frac{\partial f}{\partial x}\right)^2\ \mathrm{var}\left[x\right], \tag{3.82}$$

the variance of the square-root can be opened up:

$$\mathrm{var}\left[\mathrm{std}\left[r\right]\right] \cong \exp\left[r\right]^2\ \left(\frac{\partial\sqrt{y}}{\partial y}\right)^2\ \mathrm{var}\left[y\right] = \exp\left[r\right]^2\ \left(\frac{1}{2\sqrt{y}}\right)^2\ \mathrm{var}\left[y\right] = \frac{\exp\left[r\right]^2\ z}{4y}. \tag{3.83}$$

The counting relative standard deviation of the nuclear data standard deviation is also required:

$$\mathrm{rstd}\left[\mathrm{std}\left[r\right]\right] = \frac{\sqrt{\mathrm{var}\left[\mathrm{std}\left[r\right]\right]}}{\mathrm{std}\left[r\right]} = \sqrt{\frac{\mathrm{var}\left[\mathrm{std}\left[r\right]\right]}{\mathrm{var}\left[r\right]}} = \sqrt{\frac{\exp\left[r\right]^2\ z}{4y\exp\left[r\right]^2\ y}} = \frac{\sqrt{z}}{2y}. \tag{3.84}$$

Due to the outer relative sense of rstd []:

$$\mathrm{rstd}\left[\mathrm{std}\left[r\right]\right] \equiv \mathrm{rstd}\left[\mathrm{rstd}\left[r\right]\right] = \frac{\sqrt{z}}{2y}. \tag{3.85}$$

since any multiples of the inner term are divided. For completeness, the variance of the variance is found:

$$\mathrm{var}\left[\mathrm{var}\left[r\right]\right] = \mathrm{var}\left[\exp\left[r\right]^2\ y\right] = \mathrm{var}\left[y\right]\ \exp\left[r\right]^4 = z\exp\left[r\right]^4. \tag{3.86}$$

3.8 Generating a multi-group cross-section library

3.8.1 Multi-group microscopic cross-sections

One requirement of deterministic neutron transport methods is the discretization of all functional dependencies of neutron energy. In performing this, one is presented with the awkward task of finding discrete-energy microscopic cross-sections for reaction x of isotope z over energy group ΔE_g ($\sigma_{z,x}(\vec{r}, E_g, t)$) whose products with the not-yet known discrete-energy fluxes ($\phi(\vec{r}, E_g, t)$) produce the same reaction rates as in the not-yet performed continuous energy

transport calculation ($\sigma_{z,x}(E,t) \times \phi(\vec{r},E,t)$). With $\phi(\vec{r},E_g,t)$ is defined as:

$$\phi(\vec{r},E_g,t) \equiv \int\limits_{\Delta E_g} dE\ \phi(\vec{r},E,t), \tag{3.87}$$

this property is satisfied by defining $\sigma_{z,x}(\vec{r},E_g,t)$ as in [Duderstadt and Hamilton, 1976, p. 289]:

$$\sigma_{z,x}(\vec{r},E_g,t) \equiv \frac{\int\limits_{\Delta E_g} dE\ \sigma_{z,x}(E,t)\ \phi(\vec{r},E,t)}{\int\limits_{\Delta E_g} dE\ \phi(\vec{r},E,t)}. \tag{3.88}$$

Essentially, $\phi(\vec{r},E_g,t)$ isn't known, so a value must be guessed in generating $\sigma_{z,x}(\vec{r},E_g,t)$. This process is erroneous unless either $\sigma_{z,x}(E,t)$ is constant over ΔE_g, $\phi(\vec{r},E,t)$ never deviates from the assumed spectrum, or ΔE_g is infinitesimally thin [Stamm'ler and Abbate, 1983, p. 24]. In practice, most $\sigma_{z,x}(E,t)$ have strong resonances within a ΔE_g, sharp depressions in $\phi(\vec{r},E,t)$ can grow with time as absorptive resonances appear from newly bred isotopes, and one is limited to reasonably finite sets of ΔE_g.

For this work, discrete-energy (or 'multi-group') cross-sections are needed for adjoint transport calculations. These calculations are always performed after a continuous-energy forward transport calculation so that the adjoint source can be constructed. It is possible, therefore, to extract $\phi(\vec{r},E,t)$ from the forward transport calculation, and then to generate $\sigma_{z,x}(\vec{r},E_g,t)$ exactly (if preservation of responses is desired, the adjoint distribution is also required). Strictly speaking, $\sigma_{z,x}(\vec{r},E_g,t)$ varies both spatially and temporally, so a multi-group cross-section generated for a spatial region at one point in operation is different from that of a different spatial region at a different point in operation. It takes around a week to generate a set of multi-group cross-sections, so it would be beneficial generate a single library in a way that minimizes the variance of cross-sections, so that it can be accurately used for variety of problems.

3.8.2 Generation procedure

This subsection largely mimics the documentation and procedure of the processing of the JEFF-3.1 evaluated nuclear data library [Santamarina *et al.*, 2009; OECD/NEA, 2011] into an ACE-formatted library [Cabellos, 2006]. Here, ENDF/B-VII.0 [Chadwick *et al.*, 2006; BNL/NNDC, 2011b] provides the evaluated nuclear data, which is processed into either continuous energy, multi-group, or covariance outputs. It must be emphasized that the presented procedure is imperfect and is born out of a brobdingnagian amount of trial and error. Whenever the value of a parameter is not obvious and a precedent doesn't exist (perhaps from [Trellue, 2010] and [Arcilla, 2007]), a value is chosen that achieves the best accuracy without breaking the library (producing invalid or unbalanced cross-sections). Often library

breakage or malperformance is not encountered until dozens of steps following a modification so that errors can seem unrelated to the modifications.

The NJOY99.364 [MacFarlane, 1994] and CRSRD-2F [Wagner *et al.*, 1994] nuclear data processing codes are used to process ENDF6-formatted evaluated nuclear data [CSEWG, 2009] into various formats depending on the intended usage. The Python script `Cross-Sections.py` automates the processing according to the script filename: `Endf2Point.py` for ACE-formatted (A Compact Endf) continuous energy data, `Endf2Multi.py` for ACE-formatted multi-group data, or `Endf2Cov.py` for ERRORR-formatted covariance data. As an example, the module sequence for each option in Figure 3.5 is walked through step-by-step, for ^{nat}C (with MAT `600`) at a temperature of `900` Kelvin.

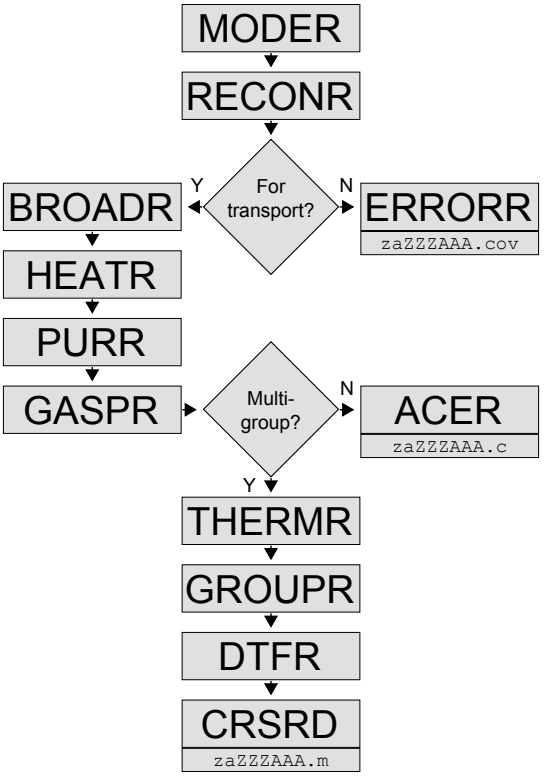

Figure 3.5: The NJOY99 module sequence for generating multi-group or continuous-energy ACE-formatted nuclear data and covariances.

First, the `MODER` module converts ENDF tape `tape20` (`20`), from ASCII to NJOY blocked binary mode tape `tape21` (`-21`).

```
moder
20 -21
```

Next, the RECONR module reconstructs point-wise cross-sections (PENDF) from resonance parameters and interpolations schemes with reconstruction tolerance 0.1% (0.001).

```
reconr
-21 -22
'6-C-Nat at 900K from ENDF/B-VII.0 njoy99.364-JES'
600 7/
0.001/
'the following reaction types are added where available'/
'    mt152  bondarenko unresolved'/
'    mt153  unresolved probability tables'/
'    mt20x  gas production'/
'    mt221  free thermal scattering'/
'    mt301  total heating kerma factor'/
'    mt444  total damage energy production'/
0/
```

If covariance outputs are desired, the ERRORR module constructs covariance matrices from cross-sections, relative or absolute covariances, and implicit relations between reactions with flat flux weighting distributed between $10^{-11} \rightarrow 20$ [MeV] and explicit relative sense (2, 19, 1E-05, 2E+07, 1). This is the final module executed when producing the zaZZZAAA.cov covariance matrix file.

```
errorr
-21 -22 0 23 0 0
600 19 2 0 1
0 0
/
1
1E-05 2E+07
```

If transport cross-sections are instead desired, the following four modules are run. First, BROADR module doppler-broadens and thins point-wise cross-sections with a thinning tolerance of 0.1% (0.001).

```
broadr
-21 -22 -23
600 1/
0.001/
900/
0/
```

Second, the HEATR module constructs point-wise heating KERMA (Kinetic Energy Release in MAterial) and radiation damage energy production cross-sections for elastic, non-elastic, and inelastic scatters, fissions, radiative captures, total kinetic KERMA, and total damage energy (302, 303, 304, 318, 402, 443, 444).

```
heatr
-21 -23 -24/
600 7 0 1/
302 303 304 318 402 443 444
```

Third, the PURR module prepares probabilistic unresolved resonance-range self-shielding probability tables with 20 probability bins (20), 100 resonance ladders (100), and a Bonderenko infinite-dilution cross-section of 10^{10} (1E+10).

```
purr
-21 -24 -25
600 1 1 20 100/
900
1E+10
0/
```

Fourth, the GASPR module generates gas-production reactions for ^{1}H, ^{2}H, ^{3}H, ^{3}He, and ^{4}He.

```
gaspr
-21 -25 -26
```

If the transport data is to be continuous-energy, the ACER module generates ACE-formatted data for use with MCNP6 of ASCII type (1), with suffix .90c (.90).

```
acer
-21 -27 0 28 29
1 0 1 .90/
'6-C-Nat at 900K from ENDF/B-VII.0 njoy99.364-JES'
600 900/
/
/
```

Alternatively, if the transport cross-section is to be multi-group, the scattering thermal library (STL) is built into the cross-section file instead of being contained separately in a .##t file. In this case, the THERMR module generates incoherent free-gas thermal scattering kernels (221), with 20 equi-probable angles (20), to a tolerance of 0.1% (0.001), up to a neutron kinetic energy of 10 [eV] (10).

```
thermr
0 -26 -27
0 600 20 1 1 0 1 221 0
900
0.001 10
```

Next for multi-group cross-sections, the `GROUPR` module is used to generate self-shielded multi-group cross-sections in GENDF format, with 8th order Legendre polynomials (`8`), a Bonderenko infinite-dilution cross-section of 10^{10} (`1E+10`), a generic thermal-fission-fusion 1374 energy groups distributed between $10^{-11} \rightarrow 20\,[\text{MeV}]$, and extracting absorptive and thermal scattering cross-sections and distributions and triton production (`3`, `3 221`, `6`, `6 221`, `23`).

```
groupr
-21 -27 0 -28
600 1 0 7 8 1 1 0
'6-C-Nat at 900K from ENDF/B-VII.0 njoy99.364-JES'
900
1E+10
1374
1.0000E-05 1.0471E-05 1.0965E-05 1.1482E-05 1.2023E-05 1.2589E-05
1.3183E-05 1.3804E-05 1.4454E-05 1.5136E-05 1.5849E-05 1.6596E-05
1.7378E-05 1.8197E-05 1.9055E-05 1.9953E-05 2.0893E-05 2.1878E-05
...
1.4183E+07 1.4229E+07 1.4276E+07 1.4323E+07 1.4371E+07 1.4418E+07
1.4466E+07 1.4514E+07 1.4562E+07 1.4610E+07 1.4658E+07 1.4706E+07
1.4755E+07 1.4804E+07 1.4852E+07 1.4901E+07 1.4951E+07 1.5000E+07
2.0000E+07
3/
3 221 'free thermal scattering'/
6/
6 221 'free thermal scattering'/
23/
0/
0/
```

Penultimately for multi-group cross-sections, `DTFR` is the last NJOY99 module, which converts from GENDF to DTF format, with 1374 energy groups (`1374`) and 900 thermal scattering groups (`900`), for 47 explicit nuclear reaction edits.

```
dtfr
```

```
-28 29 -26 30
0 1 0
9 1374 50 401 1774 47 350
221 0 0
NN NNG NNN NNNN NF NF NNF NNNF NNA NNAAA NNNA NNNNA NNP NNAA NNNAA NND NNT
NNH NNDAA NNTAA NNNNN NNNNF NG NP ND NT NH NA NAA NAAA NPP NPA NTAA
NDAA NGA NPA NDA NTA NHA NAA NKHA NFKH NKKHA NDEA NDFN NFNS NDFNS
1 2 1
2 4 1
3 16 1
4 17 1
5 18 1
6 19 1
7 20 1
8 21 1
9 22 1
10 23 1
11 24 1
12 25 1
13 28 1
14 29 1
15 30 1
16 32 1
17 33 1
18 34 1
19 35 1
20 36 1
21 37 1
22 38 1
23 102 1
24 103 1
25 104 1
26 105 1
27 106 1
28 107 1
29 108 1
30 109 1
31 111 1
32 112 1
33 113 1
```

```
34 114 1
35 202 1
36 203 1
37 204 1
38 205 1
39 206 1
40 207 1
41 301 1
42 318 1
43 443 1
44 444 1
45 455 1
46 470 1
47 471 1/
/
'C-Nat' 600 1 900/
0/
```

Finally, multi-group cross-sections are processed with CRSRD2F to convert from DTF to ACE format, with the MORSE discrete angle scattering treatment (`iang = 1`) and processing of balanced and unbalanced cross-sections (`iproc = 1`).

```
za006000.dtf dtf
type1 = za006000.m
i2lp1 = 1
iskip = 0
ilen = 1774
itpos = 50
ispos = 401
ititl = 2
iengb = 1
iincp = 1 1374 1
ipn = 8
iang = 1
ibal1 = 1 1374
iproc = 1
energy
2.0000E+01 1.5000E+01 1.4951E+01 1.4901E+01 1.4852E+01 1.4804E+01 1.4755E+01
1.4706E+01 1.4658E+01 1.4610E+01 1.4562E+01 1.4514E+01 1.4466E+01 1.4418E+01
1.4371E+01 1.4323E+01 1.4276E+01 1.4229E+01 1.4183E+01 1.4136E+01 1.4089E+01
...
```

```
2.8840E-11 2.7542E-11 2.6303E-11 2.5119E-11 2.3988E-11 2.2909E-11 2.1878E-11
2.0893E-11 1.9953E-11 1.9055E-11 1.8197E-11 1.7378E-11 1.6596E-11 1.5849E-11
1.5136E-11 1.4454E-11 1.3804E-11 1.3183E-11 1.2589E-11 1.2023E-11 1.1482E-11
1.0965E-11 1.0471E-11 1.0000E-11
   materials
   6000.90 12.01103799
   edit
2 4 16 17 18 19 20 21 22 23 24 25 28 29 30 32 33 34 35 36 37 38 102 103 104
105 106 107 108 109 111 112 113 114 202 203 204 205 206 207 301 318 443 444
455 901 471
```

Several manual steps are required before a library is suitable for transport and depletion calculations, including (but not limited to) augmentation of ZAID values according to metastable conventions [Nuclear and EOS Data, 2008, p. 3], conversion from ASCII to BINARY format with `MAKXSF` [X-5 Monte Carlo Team, 2005, Appendix K], and within `XSDIR` setting the `DATAPATH` value, changing `route` to `0`, and ensuring that no rows surpass 80 columns.

3.8.3 Library customizations

A lot of time and energy was spent tweaking the multi-group energy binning to strike a balance between accuracy and remaining within memory limits. Table 3.2 summarizes the final iteration of this process–ENJEFF (Expressly Non-Joined-Energy Fhysics Files)–which contains 1374 groups, equi-lethargically spaced within an energy region (usually a decade). The general scheme was to provide more energy groups as energy increased–fine-structure resonances appear above 1 [eV] and become more narrow with increasing energy–while avoiding the usual strategy of fitting a group structure to the isotopes that are known to be important a priori. Special attention was also given to threshold and fusion energies. In the future, if memory limitations can be circumvented, or if the cross-section files can be trimmed down (they contain mostly zeros), higher resolution multi-group libraries are be feasible. Additionally, it is possible to adjust the binning density on a granularity smaller than decades, clustering groups around particularly important resonances.

Update 364 (`up364`) of NJOY [LANL/T-2, 2011] had to receive two modifications in order to successfully process nuclear data in the necessary manner. First, in continuation with `up24` and `up335`, array limits within the `DTFR` module were increased to handle up to 2641 energy groups with 50 reaction edits and 900 up-scattering groups. Second, in continuation with `up93` and [Yamamoto and Sugimura, 2006], `egrid` and `tolmin` within the `THERMR` module were adjusted to be more precise, increasing the number of elements in the former by a factor of 12 and decreasing the latter from $10^{-6} \rightarrow 3 \times 10^{-8}$. CRSRD also had to be modified in three ways. First, filename strings were expanded to 13 characters from 10. Second, the maximum number of energy groups was increased to 2641 from 300. Lastly,

Energy region [MeV]	**Groups per Decade**
$10^{-11} \rightarrow 10^{-10}$	50
$10^{-10} \rightarrow 10^{-9}$	50
$10^{-9} \rightarrow 10^{-8}$	50
$10^{-8} \rightarrow 10^{-7}$	50
$10^{-7} \rightarrow 10^{-6}$	50
$10^{-6} \rightarrow 10^{-5}$	100
$10^{-5} \rightarrow 10^{-4}$	100
$10^{-4} \rightarrow 10^{-3}$	100
$10^{-3} \rightarrow 10^{-2}$	150
$10^{-2} \rightarrow 10^{-1}$	150
$10^{-1} \rightarrow 10^{+0}$	200
$10^{+0} \rightarrow 10^{+1}$	200
$10^{+1} \rightarrow 15$	700
$15 \rightarrow 20$	1
Total energy groups	1374

Table 3.2: ENJEFF multi-group energy binning scheme.

`xsdir` record lengths `nrecnum` and `nxs` were increased from 7 to 9 digits and 6 to 8 digits, respectively. Most of these customizations are justified in turn in the following section.

3.8.4 Accuracy and generality

The most important step in gauging the accuracy of a new cross-section library is the benchmark, whereby transport results of the library in question and one deemed correct are compared. For this work, the thoroughly validated continuous energy library ENDF70 [Nuclear and EOS Data, 2008] serves as the 'correct' nuclear data. Continuous-energy libraries cannot be used for adjoint transport, which is the only transport the multi-group library will be used for, so forward transport must be used for benchmarking. Forward and adjoint transport are different (namely down-scattering in forward versus adjoint up-scattering in adjoint), so the test isn't considered to be a bounding validation, but rather a general test of the multi-group library's accuracy in modeling important nuclear phenomena.

Since ENDF70 was generated in 2008, many updates have been made to NJOY99. Therefore, ENJEFF is processed with a slightly updated code, by a different practitioner, and in a different computing environment. In order to isolate code, user, and environmental differences from multi-group/continuous-energy differences, a continuous-energy library ACEJEFF (Advanced Champion Expressly Joined-Energy Fhysics Files) is generated which shares the code, environment, and user of ENJEFF. When troublesome discrepancies arise between ENDF70 and ENJEFF, ACEJEFF helps to offer some context, showing when discrepancies are meaningful. Additionally, when it is deemed useful, results from 'worst-practice' libraries

are shown to justify certain processing decisions.

3.8.4.1 Natural carbon and thermal scattering

The first benchmark problem is a fusion source at the center of a 10 [cm] radius fully-reflective sphere filled with natural carbon at 1.1 $\left[\frac{g}{cc}\right]$. Carbon is prominent in many designs and is a light element with low absorption, so good agreement in the free elastic thermal scattering physics is necessary.

Figure 3.6 shows that ENJEFF compares extremely well with ENDF70. The flux agrees

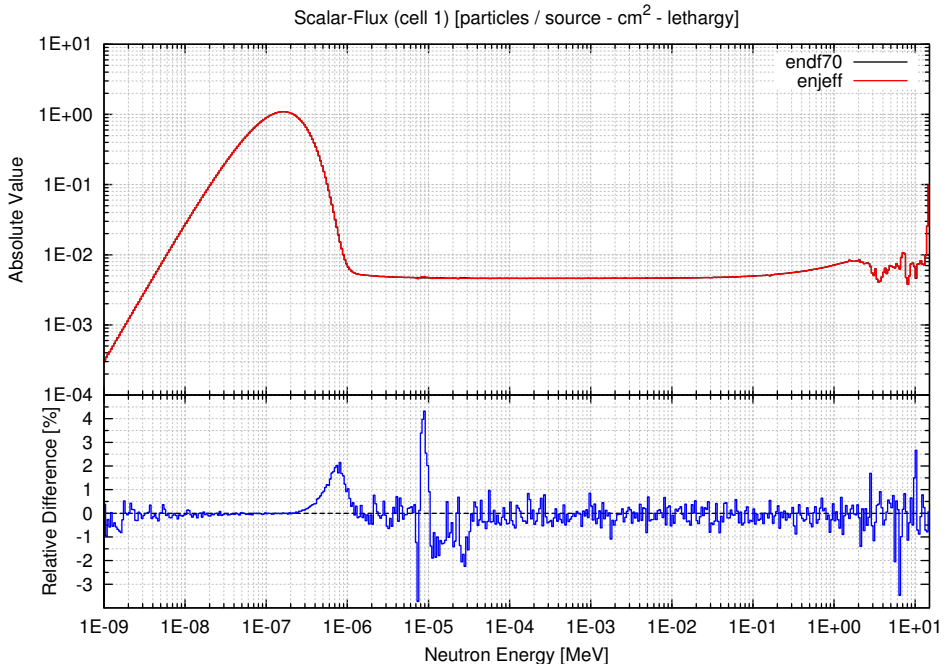

Figure 3.6: Comparison of ENDF70 and ENJEFF flux spectra for ^{nat}C.

within 5% for the entire spectrum and lays within 1% for the important thermal hump. Two small discrepancies are an oscillation at 10 [eV], where scattering physics transitions from pure down-scattering to down- and up-scattering (vector scattering to matrix scattering), and a sharp edge at the upper end of the thermal hump. They do not exist in ACEJEFF are therefore artifacts of the multi-group treatment.

It is instructive to justify certain settings and customizations employed in this work, the first of which is the inclusion of incoherent inelastic (free gas) scattering matrices with the `THERMR` module. Figure 3.7 shows the flux and radiative capture spectra when this module is turned off. Without it, the flux spectrum contains no Maxwellian thermal hump and radiative capture rates are completely off. Continuous energy MCNP6 contains internal

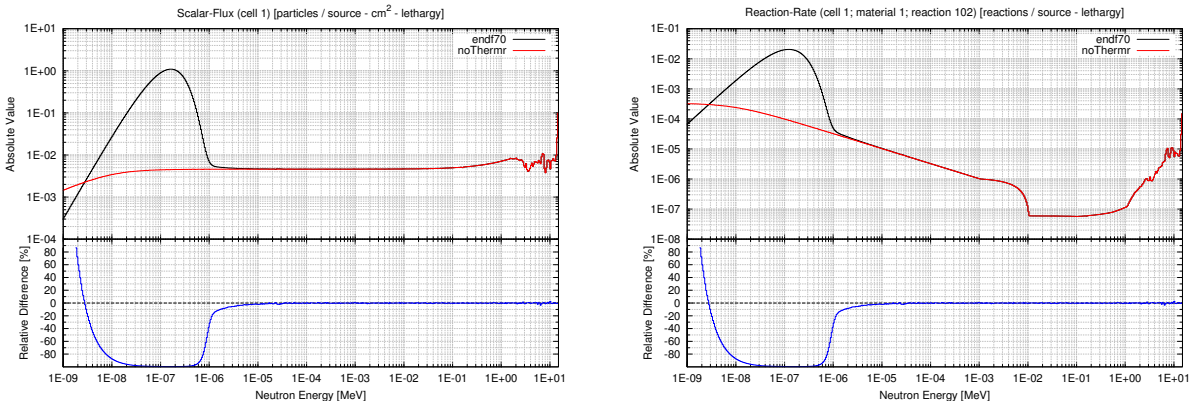

Figure 3.7: The ^{nat}C (left) flux spectrum and (right) radiative capture when the `THERMR` module is turned off.

models for free-gas physics and can also make use of coherent elastic data for crystalline materials or incoherent elastic data for hydrogenous materials, in addition to the transport data. For multi-group MCNP6, ENDF file 7 is ignored and all scattering physics must be embedded into the cross-sections [MacFarlane, 1994, p. 213],[X-5 Monte Carlo Team, 2005, p. 2-24].

Next, the `egrid` and `tolmin` edits are justified by showing the deficiencies that they avoid. Figure 3.8 shows the absorption and capture cross-sections and flux spectrum without them. NJOY conservatively derives absorption cross-sections by subtracting scattering

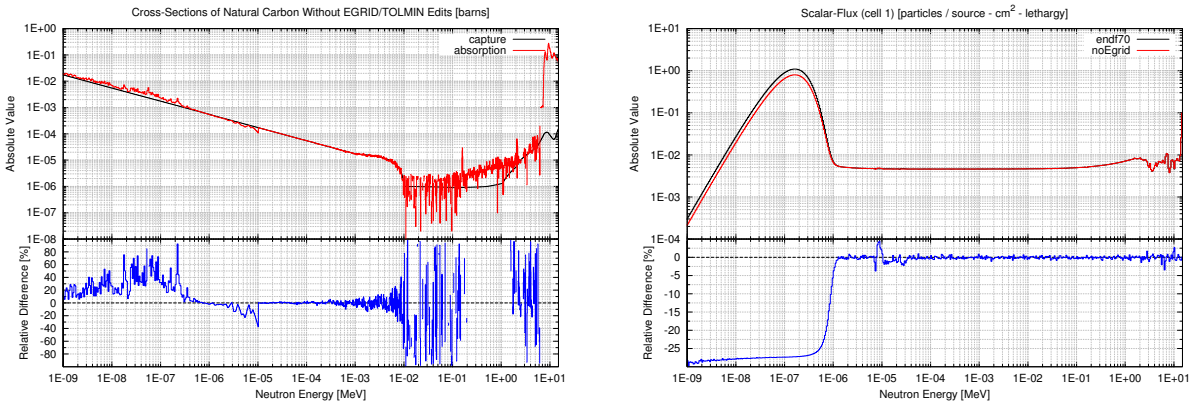

Figure 3.8: The ^{nat}C (left) absorption and capture cross-sections and (right) flux spectrum without the `egrid` and `tolmin` edits.

cross-sections from the total cross-sections. When thermal scattering physics exist, the thermal scattering matrix is numerically integrated to supply the thermal portion of the elastic

scattering vector. At low energies, radiative capture is the only absorption cross-section, so absorption and capture cross-sections should be equal. Without the edits, however, discrepancies between the cross-sections emerge at these energies (reminiscent of lin-lin interpolation of a log-log curve) and the resultant thermal flux is significantly reduced due to excess absorption. The `egrid` and `tolmin` edits effectively increase the precision to which the numerical integration is performed and avoid the erroneous thermal flux depression.

3.8.4.2 Uranium-238 and radiative capture

The second benchmark is a fusion source at the center of a 10 [cm] radius fully-reflective sphere filled with ^{238}U at 10 [g/cc]. ^{238}U is a heavy fertile/fissionable isotope, so good agreement with fine structure resonant radiative capture and threshold reactions is necessary.

Figures 3.9 and 3.10 show the progression of the neutron flux spectrum as the number of energy groups per decade is increased from 50 to 150 to that which is tabulated in Table 3.2. Gaps in the flux spectra appear at energies of 1 [eV] and below where the flux is low and

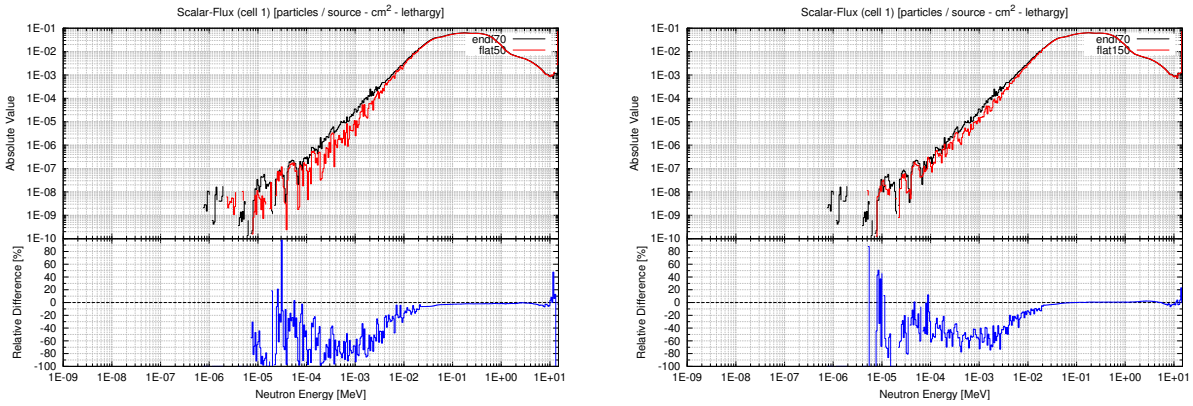

Figure 3.9: The ^{238}U flux spectra with (left) 50 energy groups and (right) 150 energy groups per decade multi-group libraries.

counting statistics are bad. This energy region is insignificant for this benchmark and can be ignored. While the flux spectra do slowly approach that of ENDF70, that of ENJEFF is inadequate; many more energy groups are necessary to achieve adequate results–possibly tens of thousands. The flux at threshold and fusion energies is within 4%, but as neutrons transition from the unresolved resonance region to the resolved resonances, capture is over-predicted and the flux falls low by 10's of percent. Radiative capture looks somewhat better but is still inadequate. Results from ACEJEFF (Figure 3.11) show similar discrepancies, but they do not exceed 10% difference until 100 [eV] and below, so blame is still cast upon the multi-group treatment.

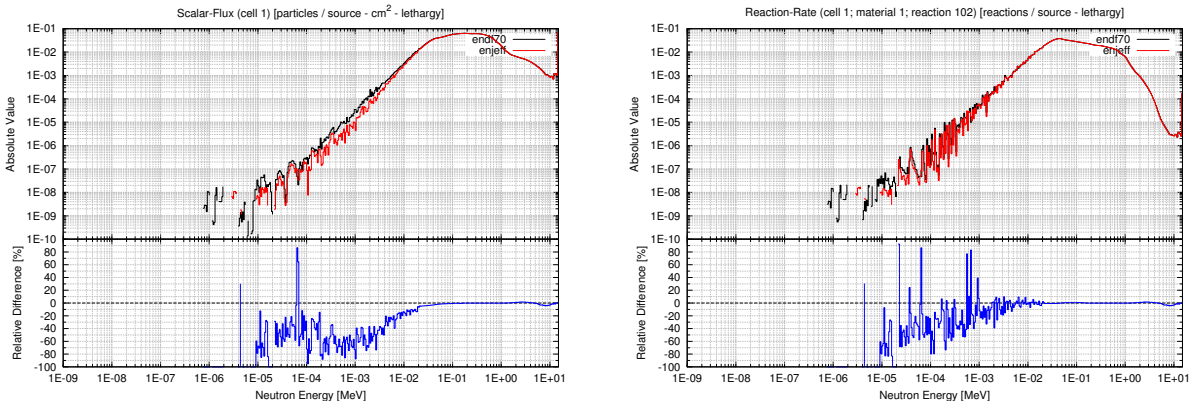

Figure 3.10: The ^{238}U (left) flux spectrum and (right) radiative capture with the ENJEFF library.

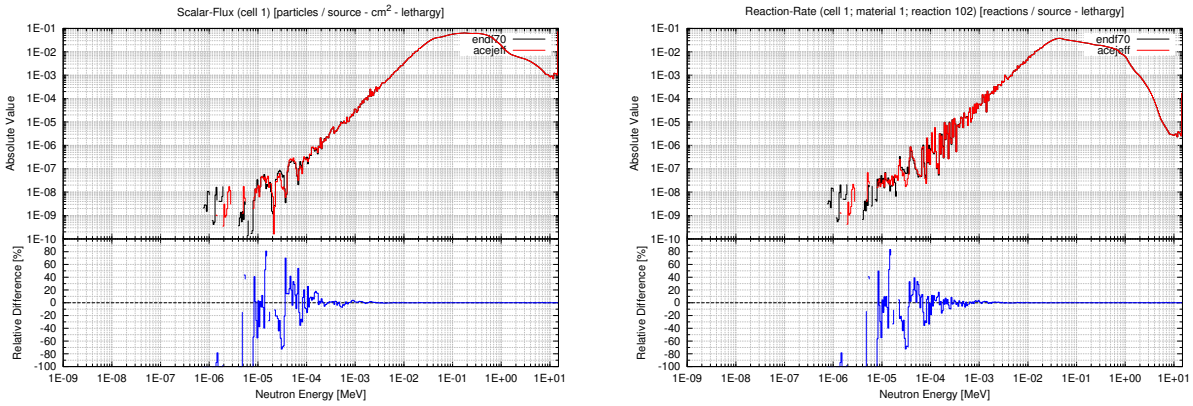

Figure 3.11: The ^{238}U (left) flux spectrum and (right) radiative capture with the ACEJEFF library.

3.8.4.3 Iron-56 and radiative capture

The third benchmark is a fusion source at the center of a 10 [cm] radius fully-reflective sphere filled with ^{56}Fe at 10 [g/cc]. ^{56}Fe is a medium-mass isotope which is the primary isotope in steel, so good agreement with fine structure resonant capture is necessary.

Like for ^{238}U, the progression of the neutron flux spectrum is shown as the number of energy groups increases (Figures 3.12 and 3.13). Whereas the resolved resonance region for ^{238}U ends at around 20 [keV], that of ^{56}Fe extends to almost 10 [MeV]. Not only does this increase the number of resonances that need to be fit, it also presents many thin (on a logarithmic scale) resonances that individually require narrow energy groups for good agreement. As a result, the isotope requires an enormous number of energy groups. Since ENJEFF is limited in its number of energy groups, it doesn't perform very well. There are large errors between 0.1 [MeV] $\rightarrow$ 2 [MeV], under-prediction of absorption in the 27.8 [keV]

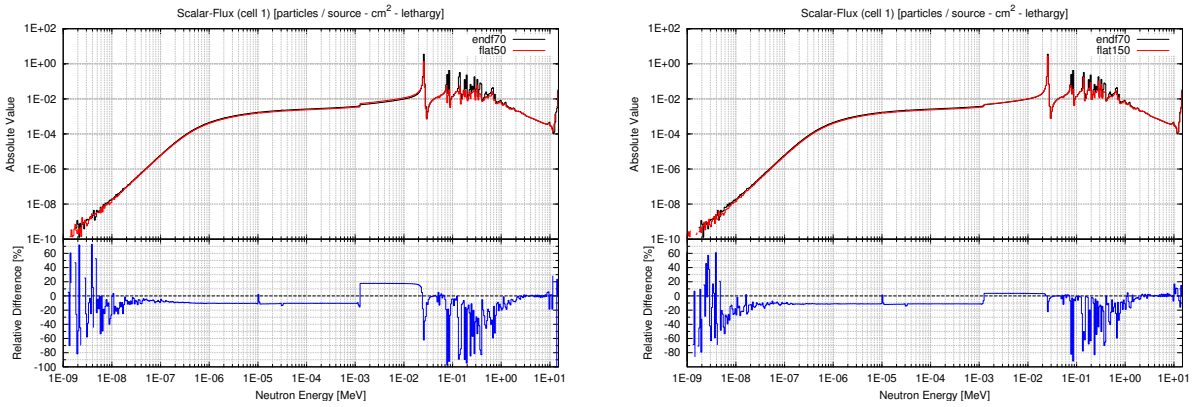

Figure 3.12: The ^{56}Fe flux spectra with (left) 50 energy groups and (right) 150 energy groups per decade multi-group libraries.

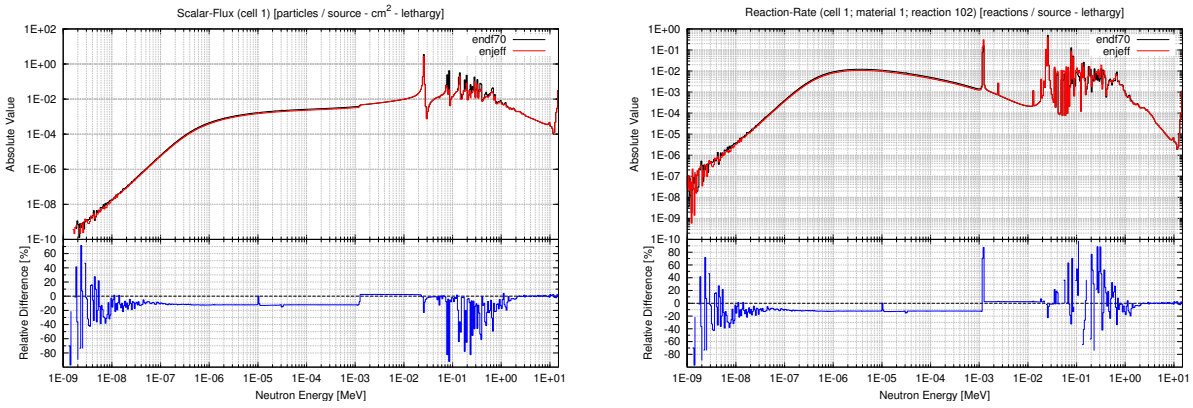

Figure 3.13: The ^{56}Fe (left) flux spectrum and (right) radiative capture with the ENJEFF library.

resonance by over 20% causes the underlying flux to be too high, and over-prediction of absorption in the 1150 [eV] resonance by almost 80% causes the underlying flux to be too low. The ACEJEFF library data is identical to ENDF70 for ^{56}Fe, so the multi-group treatment (or the insufficient number of energy groups) is solely responsible for all deficiencies.

3.8.4.4 Depleted-uranium hybrid life engine

The fourth and final benchmark is the depleted-uranium hybrid life engine (described in 4.1), which contains large amounts of the materials in the three former benchmarks. Natural carbon resides mainly within TRISO pebbles, making up 52% of the system atoms, uranium-238 is in the depleted uranium TRISO fuel kernels, making up 0.25% of the system atoms, and iron-56 is in oxide dispersion strengthened (ODS) ferritic steel structural components, making

up 0.7% of the system atoms. The benchmark is more complicated than the previous three, so a global metric was chosen for comparison of ENJEFF to ENDF70–the explicit sensitivity of tritium breeding to $^6\mathrm{Li}(\mathrm{n},\mathrm{T})$ reactions (Figure 3.14). While the ENJEFF estimate for

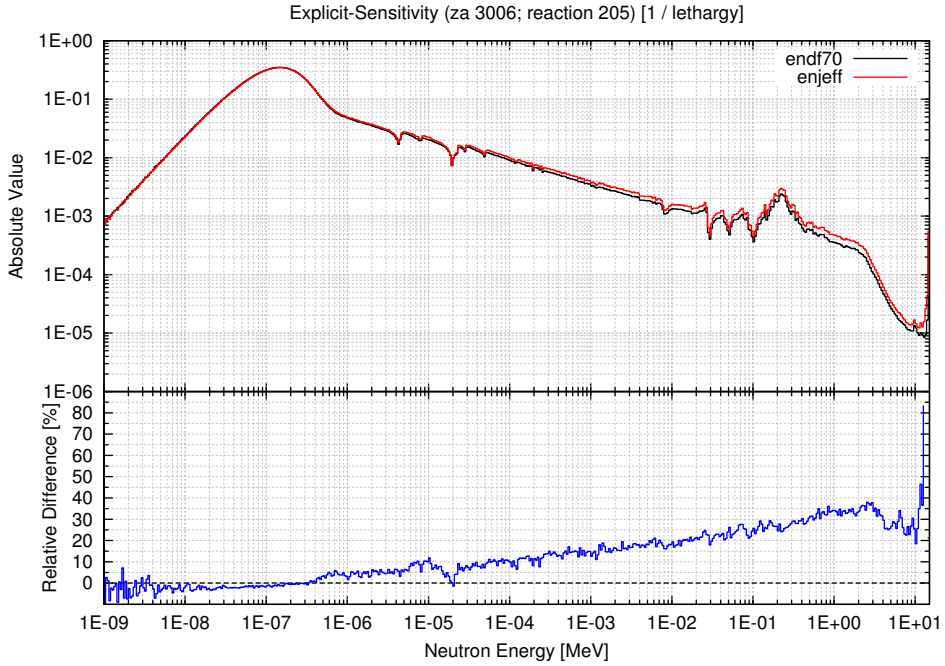

Figure 3.14: Comparison of ENDF70 and ENJEFF sensitivity of TBR to $^6\mathrm{Li}(\mathrm{n},\mathrm{T})$ reactions for DU-hybrid LIFE at BOL.

^{6}Li tritium breeding is only off by 0.08%, the estimate for total tritium breeding is off by -0.3% and the estimate for total sensitivity of tritium breeding (the integral of Figure 3.14) is off by -0.7%. These numbers aren't horrible, but they hide the gross spectral differences that exist: the sensitivity spectrum is off by 10's of percent for epithermal and fast energies. Although tritium breeding is more sensitive to thermal energies, the enormity of the error cannot be ignored.

There are a few fundamental differences between forward and adjoint transport that may be saving graces. First, adjoint sources for non-threshold and non-resonance reactions go as $\frac{1}{v}$; most adjoint particles are born at low energies. Second, adjoint elastic scatters increase particle energies instead of decreasing them; adjoint particles generally increase in energy. Consequently, most adjoint particles enter the resonance region from below, where the wider resonances reside. The tortuous path that the particles traverse will reduce the number of particles as energy increases, putting more emphasis on lower energies and less at higher energies. High energies, where discrepancies in narrow resonances and threshold reactions exist are less important than low energies, in adjoint transport. This argument falls

short of justifying ENJEFF as an accurate cross-section library for threshold and resonance responses, but should alleviate some concerns. Multi-group library generation is only a minor part of this work, so improvement is left to future work.

3.9 Microscopic cross-section covariance matrices

There are a number of excellent evaluated nuclear data libraries available. Until recently, however, little emphasis has been placed on uncertainty quantification and most libraries lack a sufficient number of covariances. Table 3.3 addresses the covariance content of nine libraries, tabulating the total number of covariance matrices and the number of isotopes for which they are distributed.

Library	**Origin (Year)**	**Size [MiB]**	**# Matrices / # Isotopes / Total**	**References**
CENDL-3.1	China (2009)	158	87 / 6 / 240	[Zhigang *et al.*, 2007; CIAE/C-NDC, 2011]
ENDF/B-VI.8	USA (2001)	108	739 / 45 / 328	[McLane and Members of the CSEWG, 1996; BNL/NNDC, 2011a]
ENDF/B-VII.0	USA (2006)	396	210 / 24 / 393	[Chadwick *et al.*, 2006; BNL/N-NDC, 2011b]
ENDF/B-VII.1‡	USA (2011)‡	800‡	1858‡ / 103‡ / 418‡	[BNL/NNDC, 2011c]
JEFF-3.1.1	EU (2009)	249	744 / 35 / 381	[Santamarina *et al.*, 2009; OECD/NEA, 2011]
JENDL-3.3	Japan (2002)	116	415 / 20 / 337	[Shibata *et al.*, 2002; JAEA/NDC, 2011a]
JENDL-4.0	Japan (2010)	572	2155 / 93 / 406	[Shibata *et al.*, 2011; JAEA/NDC, 2011b]
Low Fidelity	USA (2008)	14	1838 / 387 / 387	[Little *et al.*, 2008; DOE/NNSA, 2011]
RUSFOND-2010	Russia (2010)	293	83 / 4 / 686	[Rosatom, 2006, 2011]

Table 3.3: Availability of nuclear data covariances circa December 2011. ‡ ENDF/B-VII.1 is in beta3 and is subject to change before final release.

ENDF/B-VII.0 is the basis for this works neutron transport data, but lacks many important covariances; ENDF/B-VII.1 (which is in beta 3 at the time of this writing) is expected to partially address covariance deficiences by increasing the number of covariances to 103 isotopes, but with a future release date of December 2011, and a large number of isotopes missing, it falls short. A complete covariance library generated with respect to nominal

values is necessary. The Low Fidelity covariance library satisfies these requirements and therefore was chosen to provide all covariances by default. The library is currently the most complete set of covariances and also tends to have conservatively larger variances than other libraries. It however contains a number of deficiencies, whose remedies are discussed further.

The library lacks covariance data for a number of isotopes, so other libraries were required as supplements: ENDF/B-VII.0 provides data for ^{7}Li, JENDL-4.0 provides data for ^{232}Th, ^{233}U, ^{235}U, ^{238}U, and ^{239}Pu, and JENDL-3.3 provides data for ^{nat}V. Another complication arose due to the implicit way the Low Fidelity covariance library defines many of its covariances: e.g. elastic scattering is often defined as the total less all absorptive reactions. Addition and summation of covariances requires their absolute values, which can only be constructed with the relative covariances and their absolute cross-section counterparts. The `ERRORR` module in NJOY99 is equipped to perform all necessary operations (as well as incorporation of MF=32 resonance uncertainties) to generate explicit relative covariances when provided the right information.

The ENDF/B-VII.0 and Low Fidelity ENDF6 files were concatenated and stitched together by modification of index terms according to the ENDF6 format document [CSEWG, 2009]. Three Low Fidelity files were invalid and required edits in order to produce correct covariance files. Fields 2 and 3 on line 161 for ^{19}F had to be swapped in order for energy values to be monotonically increasing. For the same reason, field 3 on line 125 for ^{252}Cf had to be increased $18 \rightarrow 19\,[\text{MeV}]$. No edits were sufficient for successful processing of ^{232}U, so its data was supplemented by JENDL-4.0. Appendix A contains the complete list of the covariance source libraries according to isotope, with isotopes that required a pass through `ERRORR` demarcated with a ‡ symbol.

The Python script `CrossSections.py` is also able to parse ENDF6- and ERRORR-formatted covariance files to extract covariance matrices (when entitled `Endf2Cov.py`). For example, the $^{238}\text{U}(\text{n}, \text{total})$ covariance matrix parsed from JENDL-4.0 is shown in Figure 3.15. These covariance matrices are then used to populate a `DoubleEnergyDistribution` object, which can be be multiplied by an `EnergyDistribution` object in quadratic form (Equation 2.1).

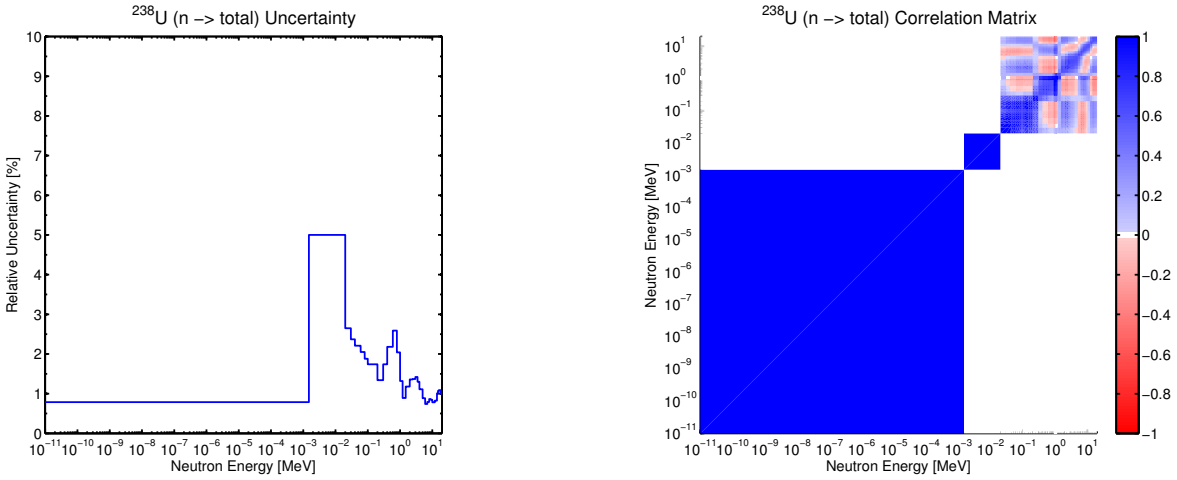

Figure 3.15: The (left) relative uncertainty and (right) correlation matrix for $^{238}\mathrm{U}(\mathrm{n},\mathrm{total})$. Covariances courtesy of [Shibata *et al.*, 2011; JAEA/NDC, 2011b].

Chapter 4

Sensitivity and Uncertainty in LIFE

In this chapter, the methods developed and described in Chapters 2 and 3 are exercised on an example nuclear system. First, the system and its important characteristics are described. Then adjoint source distributions and the subsequent adjoint/importance distributions for four neutronic responses of that system are shown and analysed for five instances during the system's life-cycle. An implicit sensitivity analysis and an uncertainty analysis are performed next. Asides are taken to quantify simplifications that are sometimes done in the literature, such as a comparison between explicit and implicit sensitivities and uncertainties and a comparison between 'full-covariance' uncertainties and 'diagonal-only' uncertainties. Statements are made regarding the general effect of nuclear data on simulations of the system, and some validation is performed for the methods.

4.1 Description of the DU-hybrid LIFE blanket

The depleted uranium hybrid LIFE blanket strives to close the fission fuel cycle without enrichment or reprocessing, while simultaneously achieving high discharge burnups with reduced proliferation concerns [Moses *et al.*, 2009; Abbott *et al.*, 2009; Kramer *et al.*, 2009; Seifried, 2009; Kramer, 2010; Fratoni *et al.*, 2010; Powers *et al.*, 2010; Kramer *et al.*, 2010; Seifried *et al.*, 2010; Kramer *et al.*, 2011; Seifried *et al.*, 2011]. Tritium and deuterium fuel the engine's DT fusion reaction. Since tritium decays with a half-life of 12.32 [y] and does not occur naturally, it must be synthesized during operation by way of neutron capture on the strong ^{6}Li tritium production cross-section shown in Figure 4.1. Carbon and a molten salt coolant 'flibe' (a mixture of LiF and BeF_2) act as moderating materials to bring fusion neutrons from their initial energy of 14.08 [MeV] down to thermal energies where they are more likely to produce tritium. A beryllium multiplier region also helps to slow and multiply fusion neutrons. The high energy of the fusion neutron source allows for efficient breeding of fissile materials as well as the fast fission of heavy metals. Fissile (and fissionable) fuel is

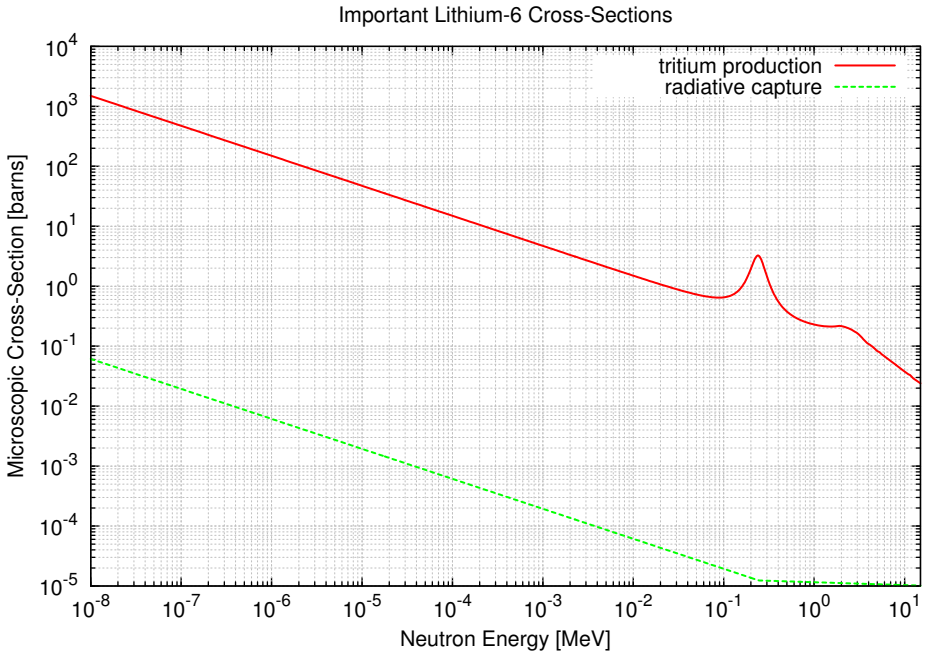

Figure 4.1: The ^{6}Li cross-section is large at thermal energies and almost exclusively produces tritium. Nuclear data is courtesy of NADS [McKinley *et al.*, 2004].

responsible for multiplication of thermal energy in the blanket at fast and thermal neutron energies.

Three critical nuclear reactions must occur throughout the lifetime of the system: tritium and fissile fuel breeding and energy release through fission of heavy metals. The figure of merit for tritium production is the tritium breeding ratio (TBR), which is defined as the number of tritons produced per source particle (every fusion reaction consumes one triton and produces one source neutron). When the TBR is greater than 1, there is a net production of tritium (ignoring radioactive decay and processing losses). The measure of fissile fuel breeding is the conversion ratio (CR), which is defined as the rate of production of fissile isotopes divided by the rate of destruction of fissile isotopes. When the CR is greater than 1, there is a net production of fissile fuel; when the CR is less than 1, there is net destruction of fissile fuel. Two measures of blanket multiplication are the thermal energy multiplication factor (M_{th}) and the neutron multiplication factor (M_n), which are defined as the ratio of thermal power output to thermal power input and the ratio of neutrons output to neutrons input, respectively. These last three neutronic responses are discussed more explicitly in Section 3.6. Together, these four performance parameters: TBR, CR, M_{th}, and M_n (the last two being somewhat redundant) form the neutronic figure of merit responses that are studied in this chapter.

The high burnup traversal of the design (see Figure 4.3) requires both that model uncertainties be accumulated over many time steps and that more reliance be placed on less certain isotopic data. The typical light-water reactor (LWR) operates across a small region of uranium enrichments and accumulates only a small fraction of the amount and variety of transuranics that are produced in the LIFE blanket. Additionally, the high energy of the fusion source shifts nuclear data dependency to energies an order of magnitude higher than an LWR. Lastly, whereas an LWR uses ordinary water for its coolant and moderator, the LIFE blanket uses carbon and flibe. All of these differences cause the expected neutron transport modeling uncertainty to exceed that of an LWR.

Control of the LIFE blanket is performed by adjusting the enrichment of lithium in the flibe coolant (see Figure 4.2); natural lithium is composed of 7.5% and 92.5% of the isotopes ^{6}Li and ^{7}Li, respectively. This single degree of freedom cannot control all operational

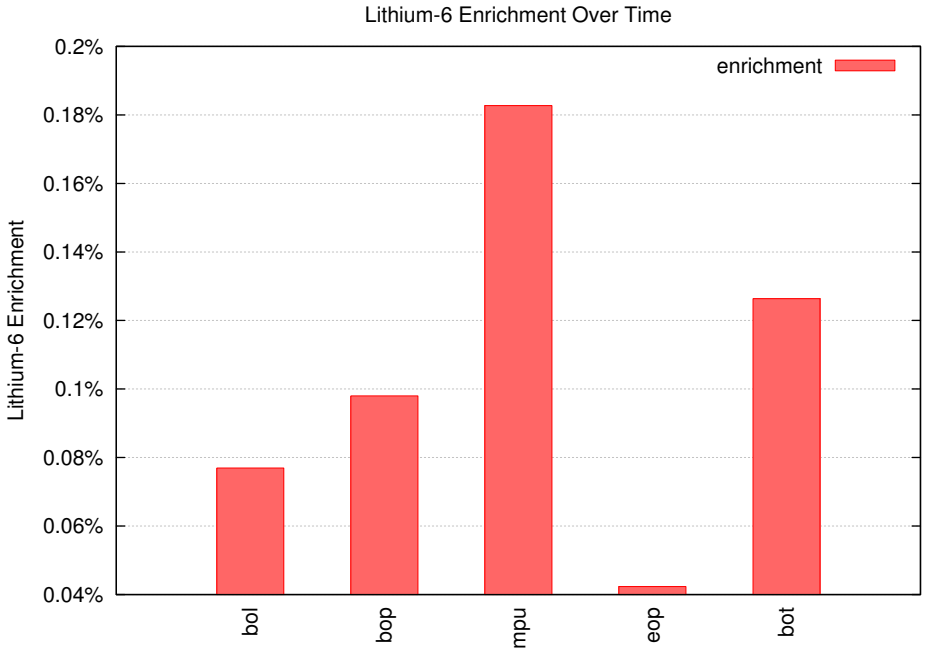

Figure 4.2: The ^{6}Li enrichment is adjusted over time for system control. The time-step title abbreviations are explained shortly.

paramaters independently and consequently, there are periods of operation in which one or more of the operational parameters have undesirable values (see Figure 4.3). The operational strategy of the LIFE blanket has three distinct phases: the breed-up phase, the power plateau phase, and the tail incineration phase. During the breed-up phase ($0 \rightarrow 2\%$ FIMA), fissile fuel is bred as quickly as possible with breakeven tritium production, at the expense of thermal power. Once the system can sustain full thermal power with breakeven tritium production,

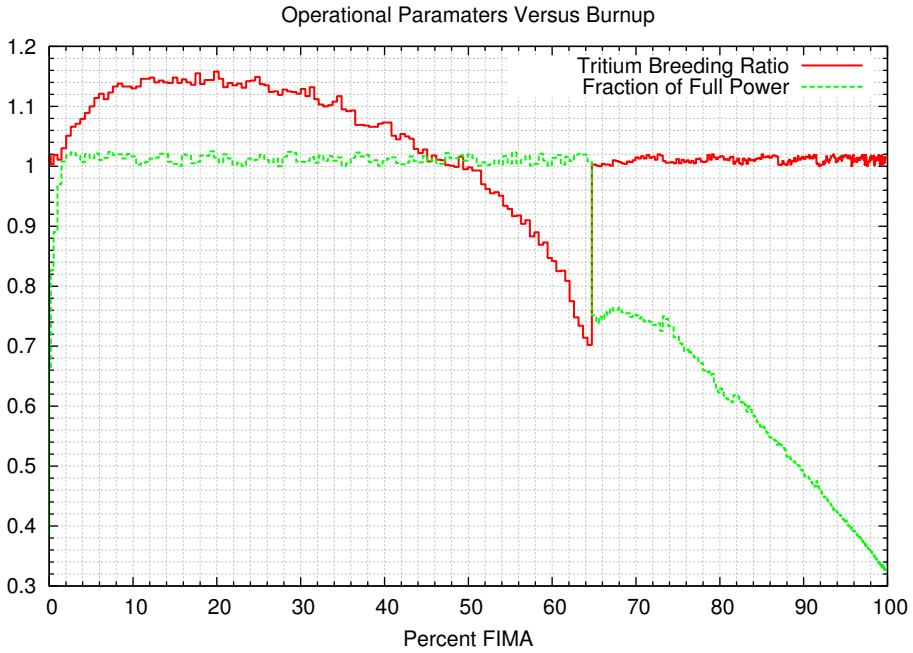

Figure 4.3: The LIFE operational parameters behave differently in each of the operational phases.

operation transitions to the power plateau phase (2 $\rightarrow$ 65% FIMA). At the beginning of this phase, tritium and fissile fuel are bred faster than they are consumed. At some point, the CR and TBR fall below one and the fissile fuel and tritium quantities start to diminish. Just before the tritium inventory is depleted, operation transitions to the tail incineration phase (65 $\rightarrow$ 100% FIMA). During this phase, the presence of fission products prevents a TBR greater than one and full power, simultaneously. Since the heart of the LIFE blanket– the fusion reaction–requires tritium, the net production of tritium is kept positive, but as small as possible at the expense of thermal power. Only the few neutrons that are leftover from tritium breeding are able to incinerate heavy metals. As the concentration of heavy metals decreases, so does the incineration rate in an asymptotic tail, which is prolonged to satisfaction.

Five instances of operation are selected for analysis because of their unique characteristics. BOL, the beginning of life, represents the system during the breed-up phase, when no fission products exist, and there are no thermal absorbers (other than ^{6}Li) and the flux spectrum is strongly thermal (see Figure 4.4). BOP is the timestep at the beginning of the power plateau phase, during which the flux spectrum is somewhat less thermal, all three operational parameters are in optimal ranges, and some fission products and transuranics are present. At MPU, the maximum amount of plutonium is present and the amount of transuranics is near its maximum. This is approximately when the CR ratio drops below unity, the TBR

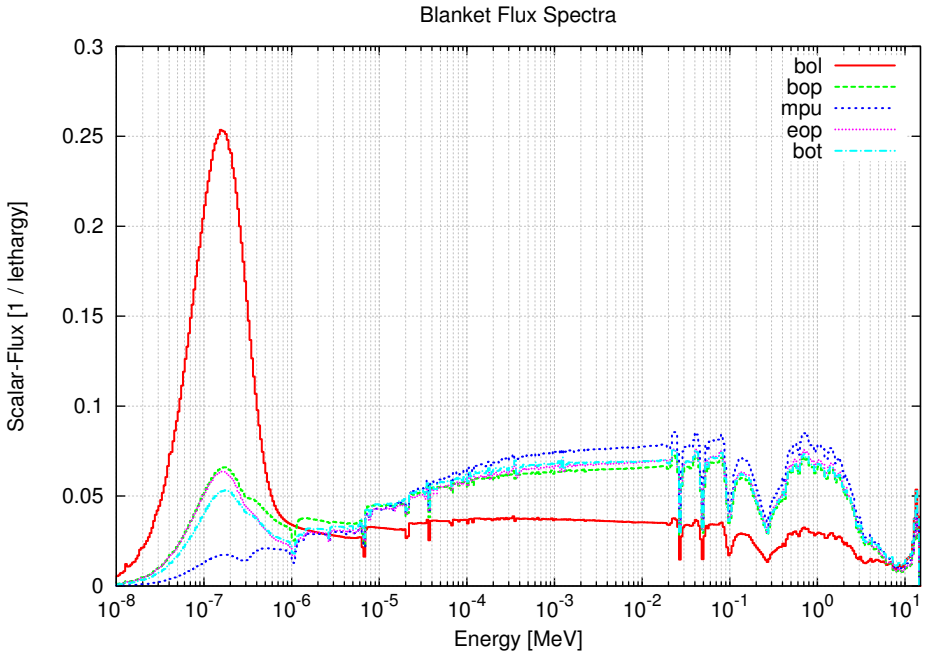

Figure 4.4: The blanket neutron flux spectrum changes strongly during the life-cycle.

peaks, and the flux spectrum is the hardest. EOP occurs at the end of the power plateau, when multiplication and TBR can no longer be sustained due to the accumulated fission products. ^{6}Li is completely removed to eliminate thermal neutron absorption by the isotope, so that multiplication of thermal energy and neutrons can be maximized. Finally, at BOT, or beginning of tail, operation is changed to the tail incineration phase. It is virtually identical to EOP, but with ^{6}Li increased for a super-unity TBR at the sacrifice of multiplication, resulting in a less thermal flux.

The version of the design discussed in this work has been largely frozen since late 2008. It consists of a 250 [cm] radius spherical fusion chamber surrounded by a series of spherical shells making up a subcritical blanket. Moving outwards from the center, the functional layers are as follows (see Figure 4.5): (1) a 25 [mm] thick tungsten armor attached to the (2) 0.275 [cm] thick oxide dispersion strengthened (ODS) ferritic steel first wall; (3) a 3 [cm] thick lithium-lead first-wall coolant; (4) a 0.3 [cm] thick ODS second wall; (5) a 3 [cm] thick flibe injection plenum; (6) a 0.3 [cm] thick porous ODS third wall; (7) a beryllium pebble multiplier region 24 [cm] in thickness; (8) a 0.3 [cm] thick porous ODS fourth wall; (8) the depleted-uranium pebble blanket 80 [cm] in thickness; (9) a 0.5 [cm] thick porous ODS fifth wall; (10) a 75 [cm] thick carbon pebble reflector; and (11) a 0.5 [cm] thick porous ODS final wall. The blanket is penetrated by 48 cones, providing an empty path for lasers to propagate to the center of the fusion chamber. Further details of the dimensions and compositions and

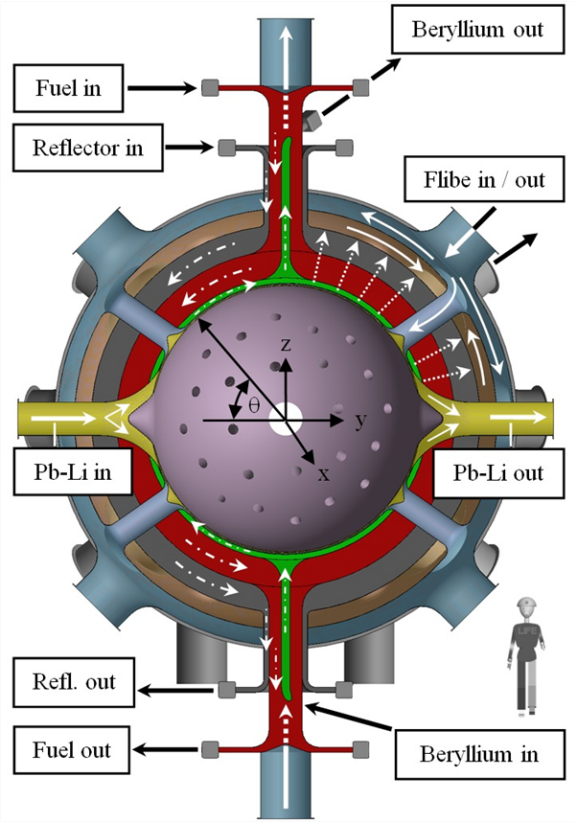

Figure 4.5: The DU-hybrid LIFE engine consists of several functional spherical shell layers. Figure is courtesy of Morris and Abbott [Abbott *et al.*, 2009].

their justifications can be found in the previously mentioned LIFE literature.

4.2 Response adjoint source distributions

As discussed in Section 3.4, adjoint sources are proportional to effective macroscopic cross-sections for a response. The units are responses per unit path length that a particle travels. It is instructive to compare adjoint sources for different spatial regions across different times to show how the capacity to cause a response changes throughout operation. Total adjoint sources can be constructed from a transport calculation–the total realized response divided by the average realized neutron path-length as shown in Equation 3.53–while adjoint source distributions can be constructed purely from material compositions and cross-sections.

4.2.1 Total response adjoint sources

In the left half of figure 4.6, the total adjoint sources for TBR is shown. These quantities

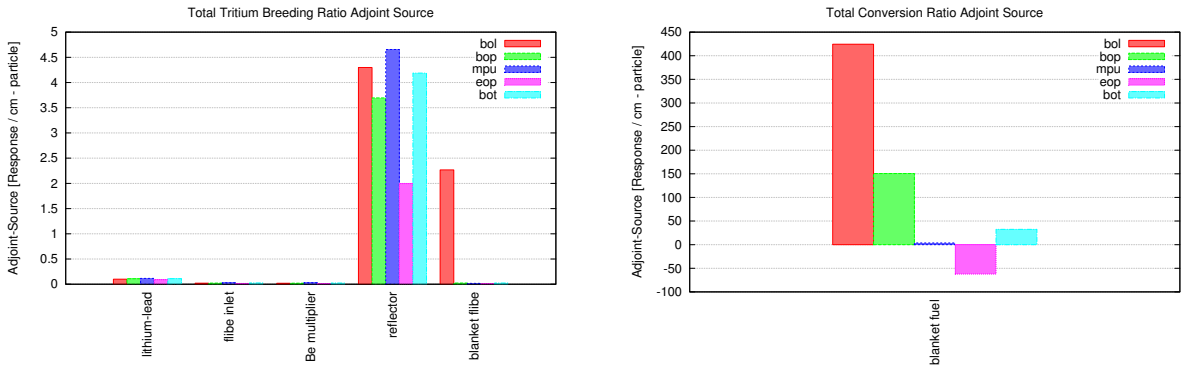

Figure 4.6: The total adjoint source for (left) TBR and (right) CR are shown over space and time.

are averaged across the entire neutron spectrum that each region contains. Variations in time are related to both the changing ^{6}Li enrichment (Figure 4.2) and the changing flux distribution (Figure 4.4). Due to the moderating carbon and flibe and absence of parasitic absorbers in the reflector, the region has around 20 times the TBR adjoint source of the other regions. This means that neutrons travelling in this region (within the systems spectrum) are 20 times more likely to produce a triton than in other regions. The lithium-lead region has roughly twice the adjoint source of the flibe and beryllium multiplier regions, which themselves have quite low total adjoint sources.

The total CR adjoint source in the right of Figure 4.6 starts large and positive, decreases with time, becomes negative around MPU, and then jumps back positive again at the end of the plateau, between EOP and BOT. The zero-crossing occurs by definition at the point in the life-cycle when fissile mass peaks. The overall decrease in adjoint source is consistent with the consistent decrease in the concentration of fertile materials in time. The jump from negative to positive is consistent with the depression of fissions due to the increase in ^{6}Li concentration.

In Figure 4.7, the total adjoint sources for M_{th} and M_n are shown. As expected, the blanket is most capable of multiplying the thermal energy and the number of neutrons, its potential to multiply in both manners peaking at MPU. The exothermic $^6\mathrm{Li}(\mathrm{n},\mathrm{T})$ and $^9\mathrm{Be}(\mathrm{n},2\mathrm{n})$ reactions are likely to occur in the reflector and beryllium multiplier region, respectively, and thus their adjoint sources are high for M_{th}. The structural regions tend to participate in radiative capture reactions, which releases energy, but typically much less than the kinetic energy of the neutron. The large exothermic $(\mathrm{n},2\mathrm{n})$ cross-sections the tungsten

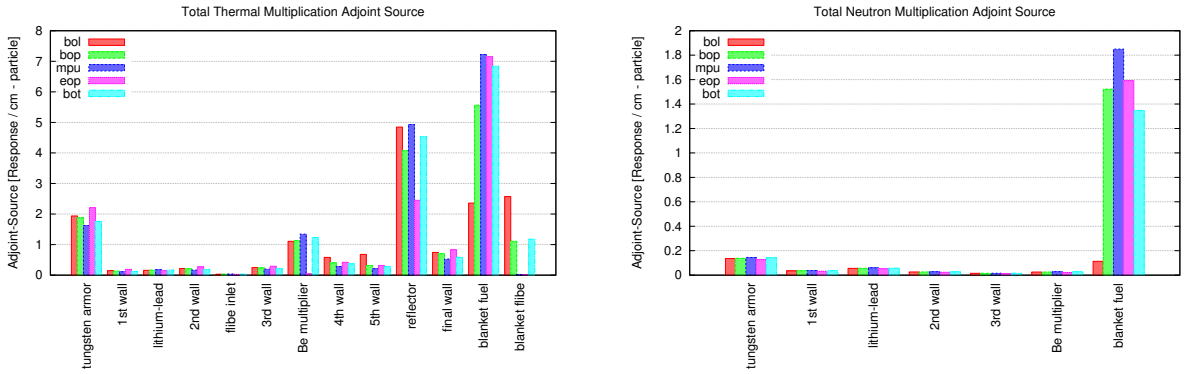

Figure 4.7: The total adjoint source for (left) M_{th} and (right) M_n are shown over space and time.

armor possesses and the hard spectrum it is exposed to grant the region a high capacity for both types of multiplication.

4.2.2 Response adjoint source energy distributions

Adjoint sources can also be plotted versus neutron kinetic energy. Whereas the former representation takes into account the physical neutron flux distribution (see Equation 3.53), this one does not. Figure 4.8 shows how the adjoint source changes between BOL and MPU, due to the changes in ^{6}Li enrichment. The + and − signs in legend entries indicate the

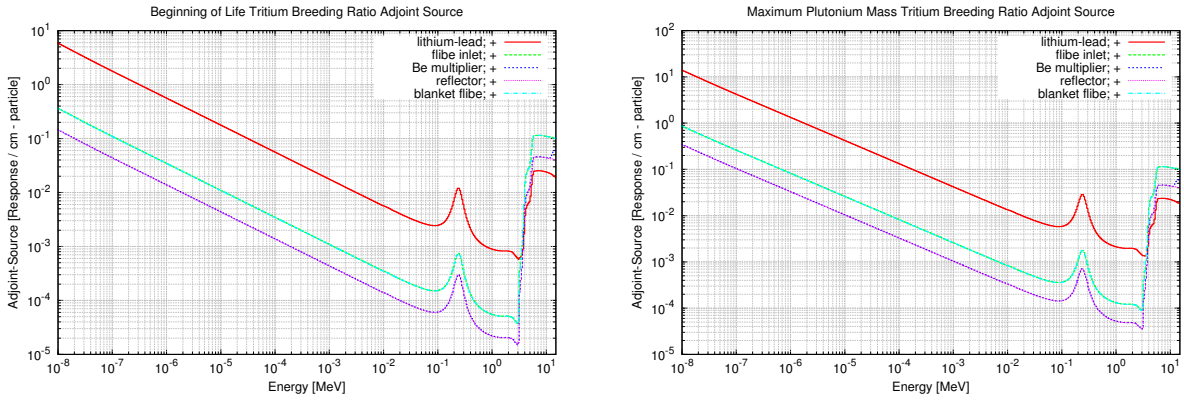

Figure 4.8: The adjoint source distribution is shown for TBR at (left) BOL and (right) MPU.

positivity or negativity of an adjoint source since logarithmic scales can display only one or the other. The lithium-lead region has the largest thermal adjoint source and the lowest fast adjoint source. However, since it sees a fast neutron spectrum, its total effective adjoint

source is only slightly higher than the other regions. The reflector adjoint source has the smallest thermal adjoint source, but ends up being larger in a total sense due to the neutron spectrum it sees.

The conversion ratio adjoint source distribution (Figure 4.9) changes a bit more drastically between BOL and MPU. Both are negative at energies less than 1 [eV], but while BOL

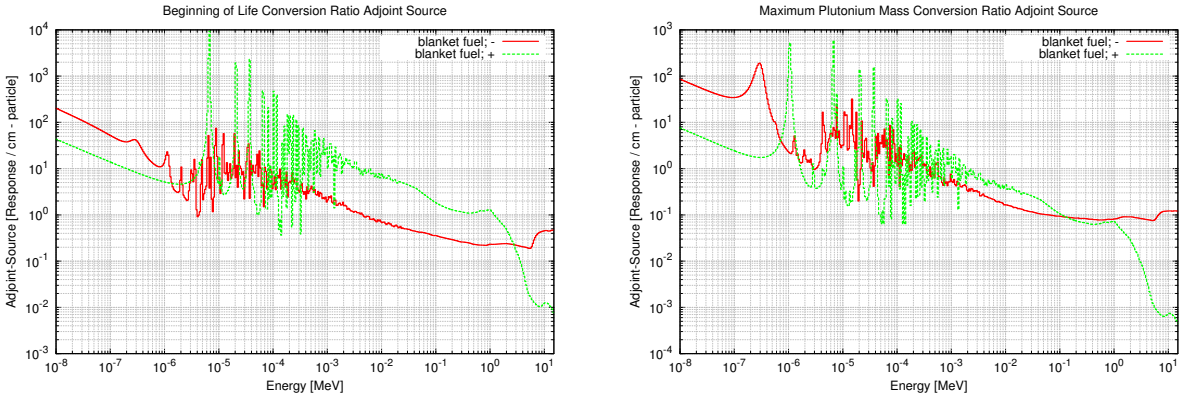

Figure 4.9: The adjoint source distribution is shown for CR at (left) BOL and (right) MPU.

is negative above 3 [MeV], MPU is negative much lower at around 130 [keV]. The 0.3 [eV] $^{239}\mathrm{Pu}(\mathrm{n}, \mathrm{fission})$ resonance is very prominent at MPU and the 6.67 [eV] $^{238}\mathrm{U}(\mathrm{n}, \gamma)$ resonance is prominent in both. In the resonance region, the adjoint source oscillates between positive and negative when fertile resonance and fissile resonances are significant, respectively.

The multiplication adjoint source distributions are shown in Figure 4.10 for BOL. For

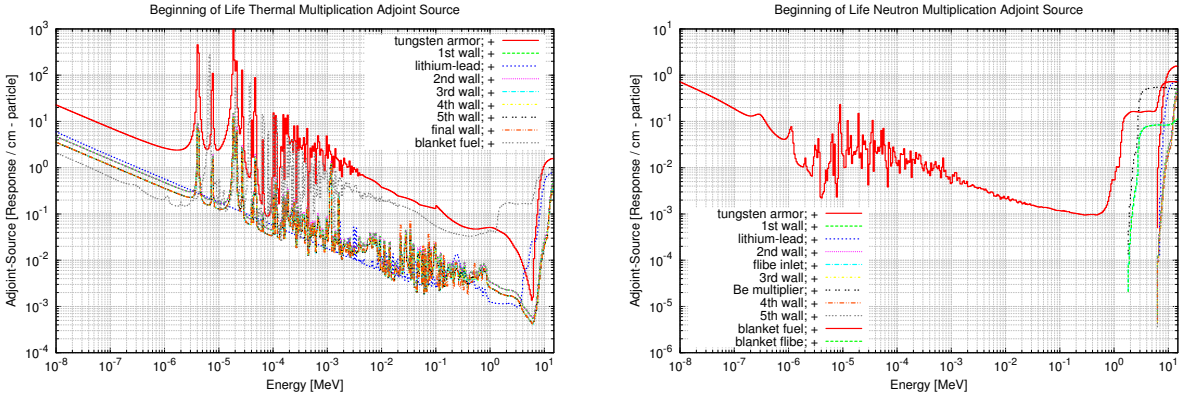

Figure 4.10: The adjoint source distribution is shown at BOL for (left) M_{th} and (right) M_{n}.

thermal multiplication, parasitic absorbers in the structures are overcome by fissions in the

blanket in both thermal and resonance regions. The exothermic reactions (such as (n, n'), $(n, 2n)$, and $(n, 2n)$) that exist above $5-6$ [MeV] boost thermal multiplication for the blanket and lithium-lead regions. For neutron multiplication, the blanket is the only region that can multiply neutrons without a threshold. Beryllium and lead dominate above thresholds for the other regions.

4.3 Response adjoint/importance distributions

As discussed in Section 3.5, importances are proportional to the number of responses incurred per neutron birth–when they are negative, neutron births decrease the response and neutron deaths bring the opposite effect. When importances are combined with knowledge of the actual rate and distribution of neutron births and deaths, the sensitivity can be found (see Section 2.4.2). By themselves, the importances can help hone intuition for a design and guide the analyst or designer in their studies. The collapsed or 'total' adjoint/importance is calculated as taking the average of a distribution over regions of neutron phase space and as such, 'total importance' is considered synonymous with 'average importance' in this work.

4.3.1 Total response importances

In the left half of Figure 4.11, the total TBR importance is shown versus space and time. In general, importances are higher at BOL, meaning that a higher TBR can be achieved

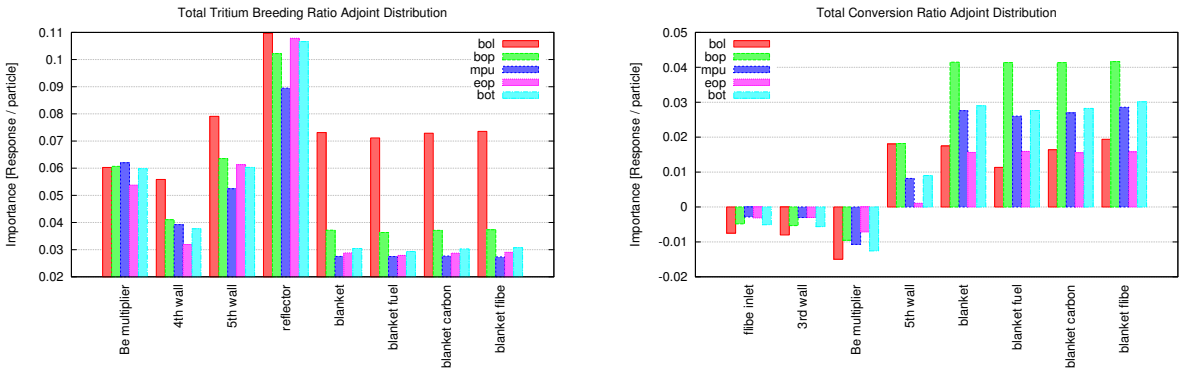

Figure 4.11: The total adjoint/importance is shown for (left) TBR and (right) CR over space and time.

with fewer neutron births. The blanket sub-regions are essentially identical. Importance is higher towards the exterior of the system (note that the reflector is physical located outside of the blanket region), where the total adjoint source is higher, except for the beryllium

multiplier, which has the opportunity to generate excess particles through (n, 2n) reactions. As thermally absorptive fission products build up in the blanket, the importance of the blanket in producing tritium decreases.

The right half of Figure 4.11 shows the total CR importance versus space and time. Importance is generally negative towards the interior and positive towards the exterior. This is because neutrons towards the interior of the blanket have to pass through many centimeters of moderator before encountering the blanket, upon which they will be thermal and more likely to fission than breed (breeding is primarily done with radiative capture in the resonance energies). Importance in the blanket peaks at BOP, drops down to bottom out at EOP, and then jumps up to an intermediate value at BOT. This oscillatory behavior is seen again in the thermal multiplication importance.

Figure 4.12, the total importance is shown versus space and time for both varieties of blanket multiplication. The blanket and its containing walls possess the highest importance,

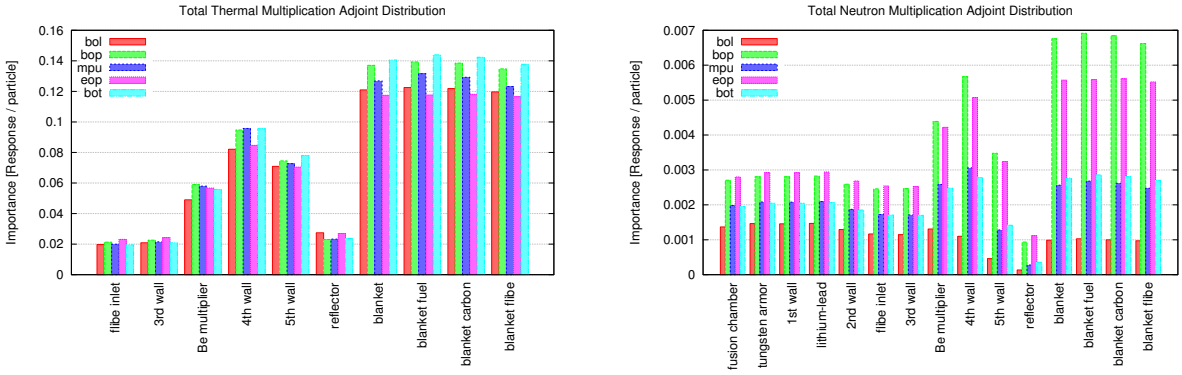

Figure 4.12: The total adjoint/importance is shown for (left) M_{th} and (right) M_n over space and time.

followed by the beryllium multiplier. The reflector is least important in terms of thermal and neutron multiplication. The time-oscillation of thermal multiplication can be explained as a combination of the accumulation of fission products and the accumulation of fissile fuel– a downward concave hump. This is the same shape as the ^{6}Li enrichment time evolution (Figure 4.2). The jump in thermal multiplication from EOP to BOT is due to the increase in the ^{6}Li enrichment, which increases the adjoint source for the exothermic ^{6}Li(n, T) reaction. At the same time, this adjustment suppresses fission reactions and thus decreases neutron multiplication importance throughout.

4.3.2 Response importance energy distributions

Adjoint/importance distributions are also useful when viewed as a function of neutron kinetic energy. In this form, they show how possible neutron source and sink spectra could be tuned to optimize a response. In Figure 4.13, the importance spectra are shown for TBR at BOL and MPU. In general, importance is larger at thermal energies for BOL, while thermal

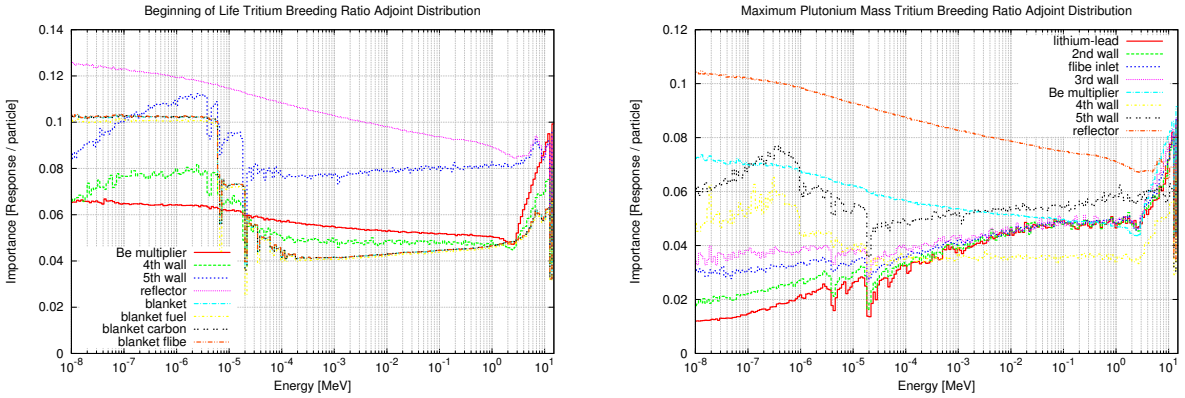

Figure 4.13: The adjoint/importance distribution is shown for TBR at (left) BOL and (right) MPU.

importances are decreased by absorptive fission products at thermal energies for MPU. The reflector has the largest importance over the entire spectrum. There are very dramatic jumps in importance within the beryllium multiplier region due to $^{9}\mathrm{Be}(\mathrm{n}, 2\mathrm{n})$ reactions. Within characteristic resonances in $^{238}\mathrm{U}$ and $^{56}\mathrm{Fe}$, there are sharp drops in importances. These resonances capture neutrons and represent a barrier of faster neutrons from reaching thermal energies where they can breed tritium.

Figure 4.14 contains the CR adjoint distributions for BOL and MPU. The spectra are negative at low energies and positive at high energies. This is because fission tends to occur at thermal energies and radiative capture breeding tends to occur within resonance and fast energies. The $^{238}\mathrm{U}$ resonances that depressed the TBR importance in Figure 4.13 are positive jumps in the CR importance. The reflector and beryllium multiplier regions, which have the utility to serve as moderators and multipliers, are positive at high energies, where they tend to push neutrons to resonance energies, and negative at low energies, where they tend to push neutrons out of resonance regions and into thermal energies.

The multiplier importance distributions at BOL are shown in Figure 4.15 below. For thermal multiplication, the blanket is most important. Multiplication is most important at high energies. This is where the cross-sections for $(\mathrm{n}, 2\mathrm{n})$ and $(\mathrm{n}, 3\mathrm{n})$ exist. For neutron multiplication, importance generally decreases outwards from the lithium-lead region to the beryllium multiplier region. The fusion chamber contains leaked importance from the neigh-

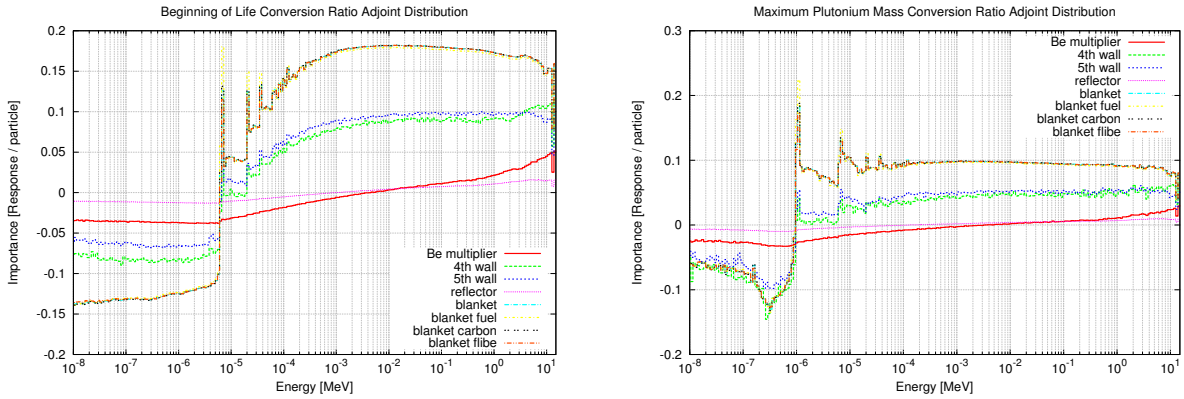

Figure 4.14: The adjoint/importance distribution is shown for CR at (left) BOL and (right) MPU.

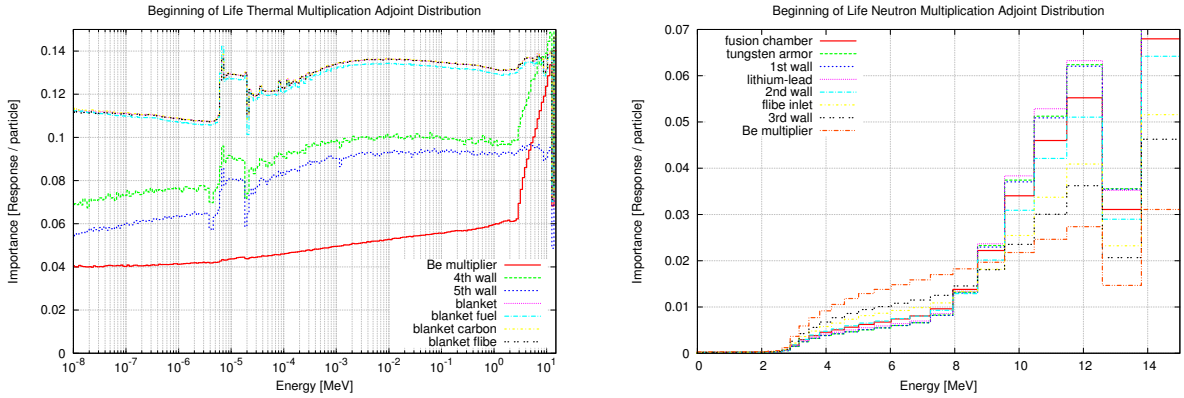

Figure 4.15: The adjoint/importance distribution is shown at BOL for (left) M_{th} and (right) M_n.

boring inner regions. The first, second, and third wall are highly important, while the more outer walls did not meet the threshold requirements for visibility on this graph.

4.3.3 Response importance angular distributions

The direction a particle travels is often important. For example, a particle located near the periphery of a system is less likely to collide and cause a response when it is heading outwards than when it is heading inwards. Because of this, adjoint/importance angular distributions are often anisotropic near geometrical features like boundaries and absorbers. Figure 4.16 shows the angular distribution (for each energy bin) of TBR importance within two spatial regions at MPU. The lithium-lead region resides towards the center of the system and is bombarded with adjoint particles somewhat isotropically. Additionally, it is the source of a portion of the adjoint source. Consequently, its distribution is somewhat isotropic. The

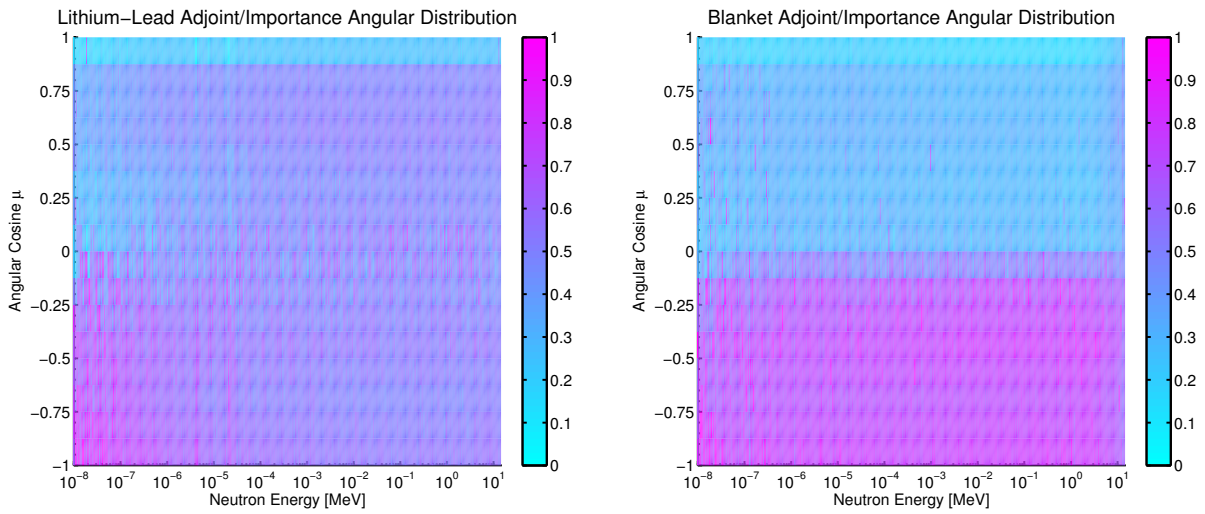

Figure 4.16: The TBR adjoint/importance angular distribution for MPU within (left) the lithium-lead region and (right) the blanket. Particles with a cosine of positive one are travelling outwards and particles with a cosine of negative one are travelling inwards.

only anomaly (one that forbids separability of angular dependence from spatial and energy dependence) is that importance at low energies preferentially heads inwards. The blanket is a different matter; its importance angular distribution is quite anisotropic. The blanket is located towards the extremity of the design and accordingly, particles that travel inwards are somewhat more important than those travelling outwards.

4.4 Sensitivity analysis

For some, sensitivities are the end of the line–the sought after result of a sensitivity analysis. Sensitivities possess none of the limitations of the metrics shown and discussed in the previous sections. Adjoint source distributions (Figures 4.8, 4.9, and 4.10) represent the capacity of a material to contribute to a response, but speak nothing of the neutron spectrum or the geometrical arrangement of materials. Total adjoint sources (Figures 4.6 and 4.7) incorporate the former, but not the latter. Total adjoint/importances (Figures 4.11 and 4.12)–the results of adjointly transporting adjoint sources within the system geometry–show the smeared average of importance over a spatial region without regard to the distribution of neutron sources and sinks. Adjoint/importance distributions (Figures 4.13, 4.14, and 4.15) are uncollapsed and therefore allow for detailed study, but still do not take into consideration the neutron sources and sinks. Only after they are considered by way of constructing the adjoint bilinear functional (see Section 3.2) is the sensitivity found.

Sensitivities quantify the linear relative change in a response with respect to a relative

perturbation in an input parameter (see Section 2.3). Explicit sensitivities discount the effect of any perturbation upon the flux, holding it constant. The more physical implicit sensitivity considers the effect of any perturbation upon the flux, to first order. Consequently, explicit sensitivities for responses due to most input perturbations are zero (many inputs are not explicitly related to a given response) and in contrast, implicit sensitivities of responses to any input (for which flux perturbations are expressly considered) are usually non-zero.

Response sensitivities sum up to one of two values: sensitivities for linear functional responses like TBR, M_{th}, and M_n add up to one; sensitivities for linear function ratio responses like CR add up to zero. For clarity of presentation, lesser sensitivities–bins for which the sensitivity is below a certain threshold–are removed from graphs. Consequently, the grand total of sensitivities for the following graphs may not add up conservatively.

4.4.1 Comparison of implicit and explicit sensitivities

The left half of Figure 4.17 compares total implicit and explicit sensitivities for TBR at MPU to isotopic reaction rates. While ^{56}Fe, ^{238}U, and ^{239}Pu do not explicitly participate

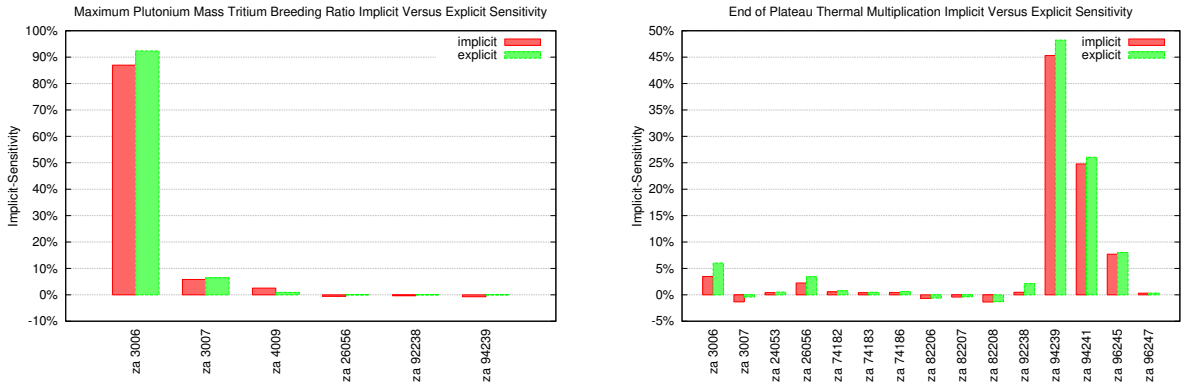

Figure 4.17: Comparison of total implicit and explicit sensitivities for (left) TBR at MPI and (right) M_{th} and EOP.

in the production or destruction tritons, their presence implicitly eliminates neutrons that could otherwise breed tritium; the explicit sensitivity to their concentrations is zero and the implicit sensitivity to their concentrations is negative. ^{6}Li and ^{7}Li breed tritium, but doing so removes neutrons from the system; their explicit sensitivities are positive and their implicit sensitivities are slightly smaller. ^{9}Be produces 0.9% of tritons at MPU, so its explicit sensitivity is 0.9%. However, $^{9}\mathrm{Be}(\mathrm{n}, 2\mathrm{n})$ reactions also generate excess neutrons that can eventually produce tritium, so its implicit sensitivity is somewhat higher (2.5%).

The right half of Figure 4.17 compares total implicit and explicit sensitivities for M_{th} at EOP and its story somewhat is similar to TBR total sensitivities at MPU. Isotopes that

participate in exothermic nuclear reactions (^{6}Li, ^{56}Fe, ^{238}U, ^{249}Pu, ^{241}Pu, ^{245}Cm, and ^{247}Cm), exhibit positive explicit sensitivities to thermal multiplication; their implicit sensitivities are also positive, but less so since their presence depresses neutrons and their ability to participate in other thermal multiplication. Thermal multiplication is explicitly negatively sensitive to isotopes that participate in endothermic nuclear reactions (^{7}Li and $^{206-208}$Pb); it is implicitly slightly more negatively sensitive to those same isotopes because they the reactions parasitically absorb neutrons.

4.4.2 Total sensitivities

Sensitivities can be collapsed and distributed over a number of parameters according to the analysis. When they are collapsed over all but cells, they represent the spatial variation of sensitivity in a design. When they are collapsed over all but isotope, they represent the sensitivity of responses to those isotopic number densities. When they are collapsed over all but reaction type, they show how different reactions participate in causing a response. When they are collapsed over all but cells and isotopes, they show how the spatial variation of isotopic number densities affect a response. This can be especially useful for optimization of responses [Vujic *et al.*, 2003]. When they are collapsed over all but cells and reaction types, they show how different reactions affect a response spatially. When they are collapsed over all but isotope and reaction type, they quantify the influence of each isotope's reaction upon the response. This collapsing scheme is appropriate when propagating nuclear data uncertainties for the reactions of certain isotopes. When sensitivities are not collapsed at all, they show how isotopic reactions affect a response versus space. This most detailed binning of sensitivities offers the analyst a rich set of information with which to garner insight for a design.

The spatial distribution of neutron multiplication sensitivities are shown over time in the left half of Figure 4.18. At BOL, when the blanket contains all fertile heavy metals and no fissile materials, it contributes only 5.9% of sensitivity to neutron multiplication while the lithium-lead region and beryllium multiplier contribute 19% and 67%, respectively. After the blanket has had a chance to breed fissile materials, it jumps to around 30% of neutron multiplication, reducing the lithium-lead and beryllium multiplier regions to around 14% and 50%, respectively. In spite of their parasitic absorption, structures generally contribute slightly positively to neutron multiplication. The right half of Figure 4.18 distributes the same sensitivities over isotope (note the reverse order of timesteps), showing that ^{9}Be is the king of multiplication, followed by ^{239}Pu, ^{241}Pu, and $^{208-206}$Pb. The same sensitivities are finally shown distributed over reaction type in the left half of Figure 4.19, for intuitive results. $(\mathrm{n}, 2\mathrm{n})$ is responsible for the lions share of neutron multiplication throughout the lifetime, with a dip during the power plateau when the blanket can sustain ample fissions.

The right half of Figure 4.19 shows the sensitivity of TBR to various reaction types. Intuitively, tritium production is almost 100% sensitive to (n, T) reactions throughout the

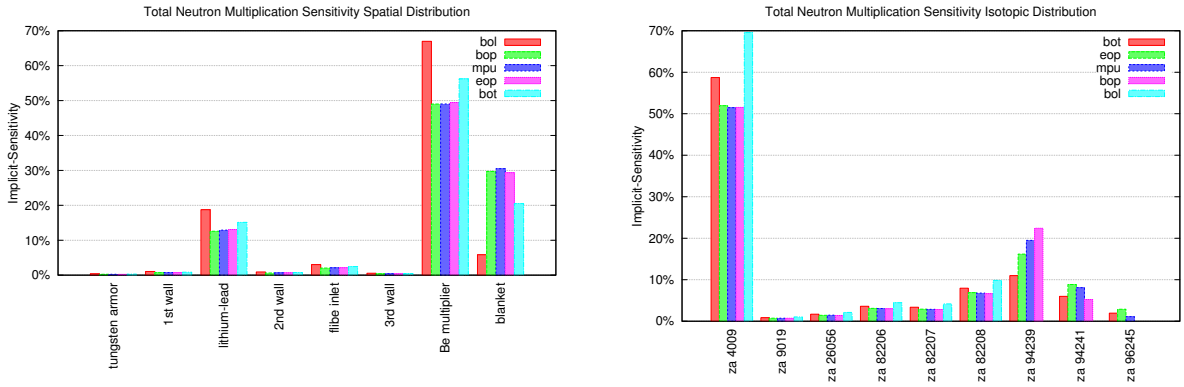

Figure 4.18: Total sensitivities for M_n distributed (left) spatially and (right) isotopically, over time.

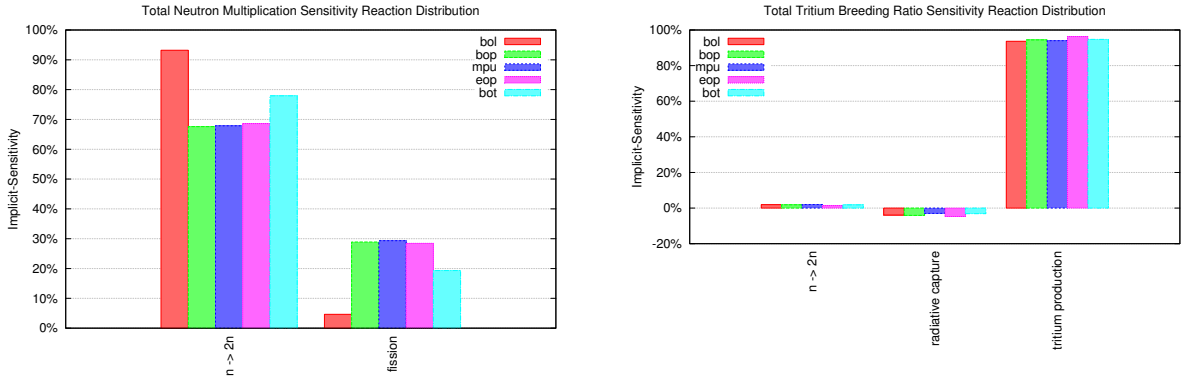

Figure 4.19: Total sensitivities for (left) M_n and (right) TBR distributed over reaction type and over time.

life-cycle, since this reaction produces 100% of tritons. TBR is slightly positively sensitive to $(n, 2n)$ reactions and doubly negatively so to parasitic capture through (n, γ) reactions.

Figure 4.20 shows somewhat intuitive total sensitivities for M_{th} distributed over spatial region and reaction type. The blanket, which participates in fewer multiplying reactions than the lithium-lead region or beryllium multiplier, releases many times more energy per reaction. The Q-value of fissions are around $+200\,[\text{MeV}]$, while those of $^{6}\text{Li}(n, T)$, $^{9}\text{Be}(n, 2n)$, $^{206-208}\text{Pb}(n, 2n)$ are $+4.78\,[\text{MeV}]$, $-1.57\,[\text{MeV}]$, $-8.08\,[\text{MeV}]$, $-6.74\,[\text{MeV}]$, and $-7.36\,[\text{MeV}]$, respectively, so the energy consumed by multiplying neutrons in the lithium-lead and beryllium multiplier regions is largely counteracted by the energy released by $^{6}\text{Li}(n, T)$ reactions. This effect can be seen in the right half of Figure 4.20, which shows equal but opposite trends in $(n, 2n)$ and (n, T) reaction M_{th} sensitivities over time. In fact, the number of $(n, 2n)$ re-

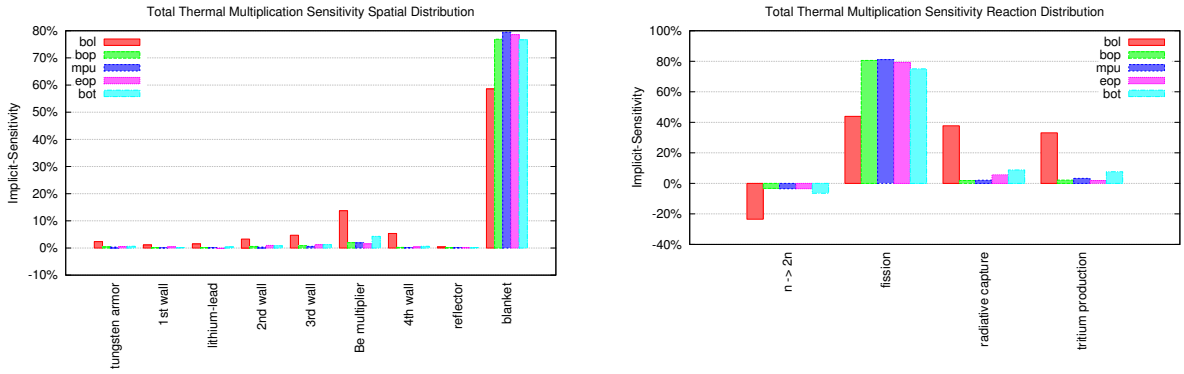

Figure 4.20: Total sensitivities for M_{th} distributed (left) spatially and (right) over reaction type, over time.

actions isn't decreasing, only its relative sensitivity, as blanket fission increases. Thermal multiplication is approximately equally sensitive to radiative capture and tritium production throughout life. The two are also responsible for 38% and 33% of sensitivity respectively, compared to fission's 44% at BOL.

4.4.3 Sensitivity energy distributions

Like the previously discussed metrics, sensitivities offer a rich abundance of information when they are distributed versus neutron kinetic energy. They can declare whether a system is 'fast' or 'thermal,' by quantifying the fraction of sensitivity present in different energy regions. They can show the effect of the spatial change of neutron spectrum spatially on a response. They can be used to propagate energy-dependent nuclear data uncertainties to response uncertainties. In general, they offer another realm for the analyst to investigate in their study of a design. Because logarithmic scales can span only positive or negative values, positive and negative sensitivities are split into their own graphs.

Figure 4.21 shows the sensitivity energy distributions for TBR at BOL for various isotopic reaction types. The sensitivity is largest for $^{6}\mathrm{Li}(\mathrm{n},\mathrm{T})$ at the thermal hump. $^{7}\mathrm{Li}(\mathrm{n},\mathrm{T})$ and $^{9}\mathrm{Be}(\mathrm{n},2\mathrm{n})$ are the second largest at energies above their thresholds–$3-4$ orders of magnitude higher than that of $^{6}\mathrm{Li}(\mathrm{n},\mathrm{T})$ in the same energy range. $^{56}\mathrm{Fe}(\mathrm{n},\gamma)$ and $^{238}\mathrm{U}(\mathrm{n},\gamma)$ take away around 1% of TBR sensitivity within resonance regions.

Figure 4.22 shows the sensitivity energy distributions for M_{th} at MPU for various isotopic reaction types. $^{239}\mathrm{Pu}(\mathrm{n},\mathrm{fission})$, $^{241}\mathrm{Pu}(\mathrm{n},\mathrm{fission})$, $^{245}\mathrm{Cm}(\mathrm{n},\mathrm{fission})$, $^{6}\mathrm{Li}(\mathrm{n},\mathrm{T})$, and $^{238}\mathrm{U}(\mathrm{n},\mathrm{fission})$ contribute the largest to thermal multiplication sensitivity. The first three are essentially distributed evenly over fast, resonance, and thermal energy regions. The plutoniums' thermal resonances are very prominent on the thermal hump. The fourth reaction

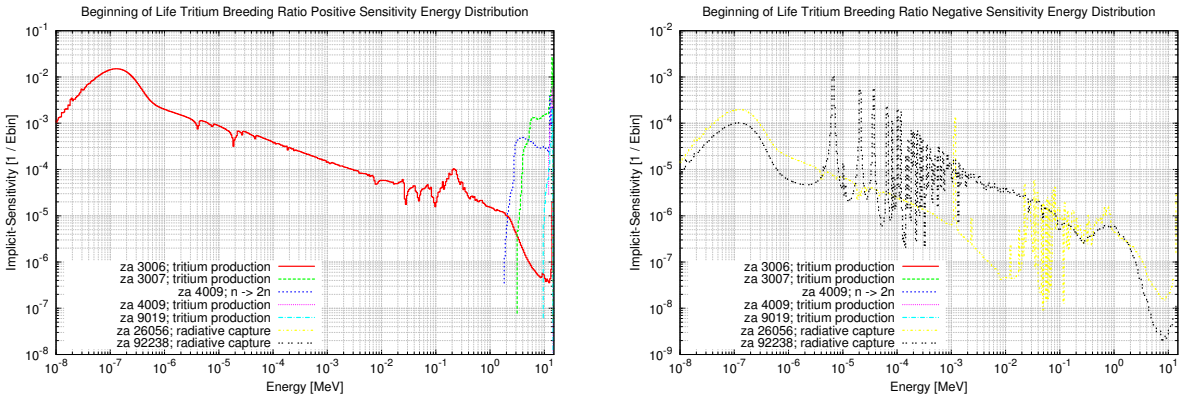

Figure 4.21: (Left) positive and (negative) sensitivity energy distributions for TBR at BOL versus isotopic reaction type.

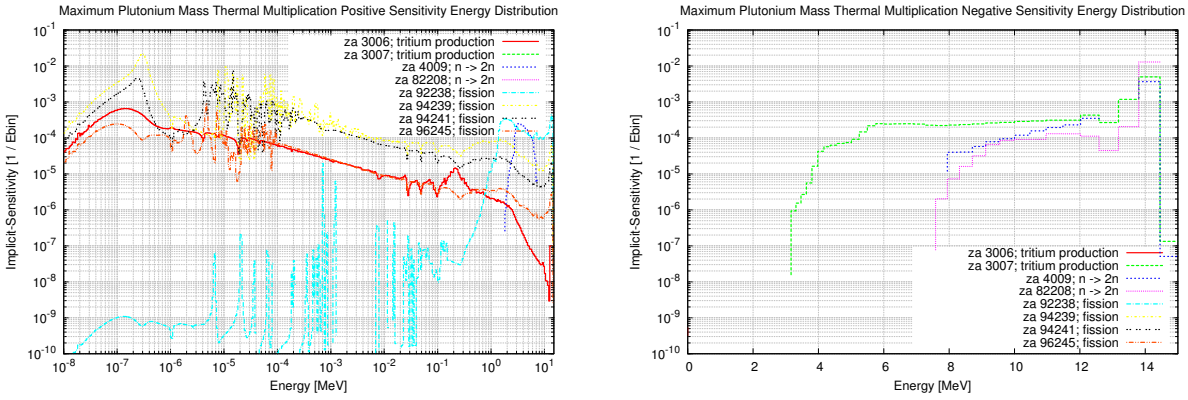

Figure 4.22: (Left) positive and (negative) sensitivity energy distributions for M_{th} at MPU versus isotopic reaction type.

contributes mostly at thermal energies and the fifth reaction contributes its majority above its threshold. The endothermic ^{7}Li(n, T), ^{9}Be(n, 2n), ^{208}Pb(n, 2n) threshold reactions are the largest energy removals from the system. At the same time, the ^{9}Be(n, 2n) is also a positive contributor to thermal multiplication sensitivity below around 8 [MeV]. This helps illustrate the complexity of implicit sensitivity and the many factors that go into such an analysis.

The TBR sensitivity energy distributions for BOP are shown in Figure 4.23, binned versus spatial region. ^{7}Li and ^{9}Be produce tritium with their threshold reactions. Beryllium and lead multiplication help to produce excess neutrons for tritium breeding at high energies. The ^{6}Li(n, T) resonance at 0.24 [MeV] is visible in all spatial regions. The thermal ^{6}Li(n, T) reaction interacts with the thermal hump quite strongly for a large contribution to sensitivity. Dips in the blanket sensitivity can be seen for ^{239}Pu and ^{238}U resonances.

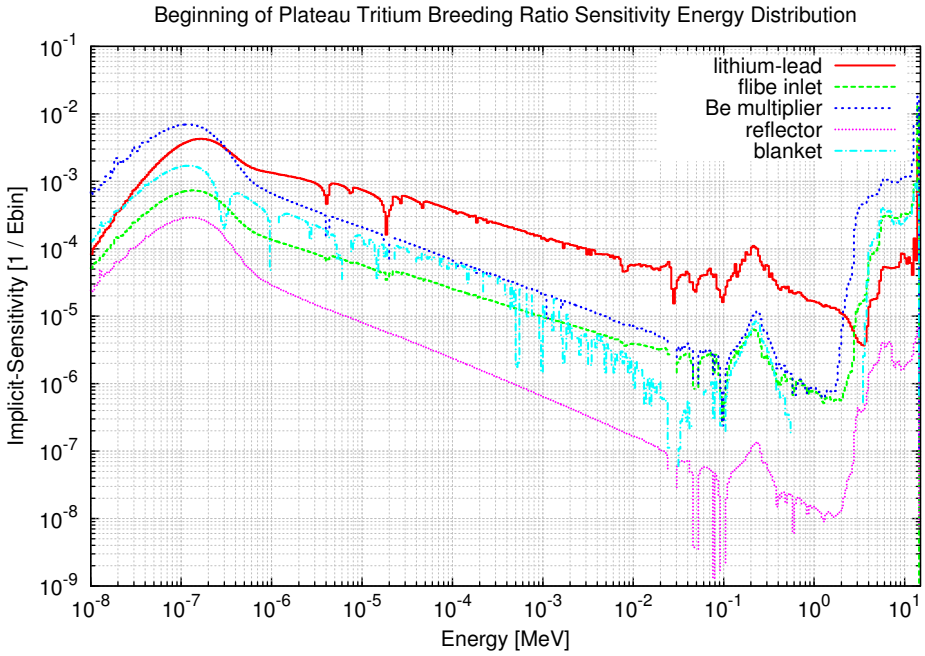

Figure 4.23: Sensitivity energy distributions for TBR at BOP versus spatial region.

The BOP M_n sensitivity energy distributions are shown in Figure 4.24, binned versus spatial region. The blanket provides most of its multiplication sensitivity at thermal and

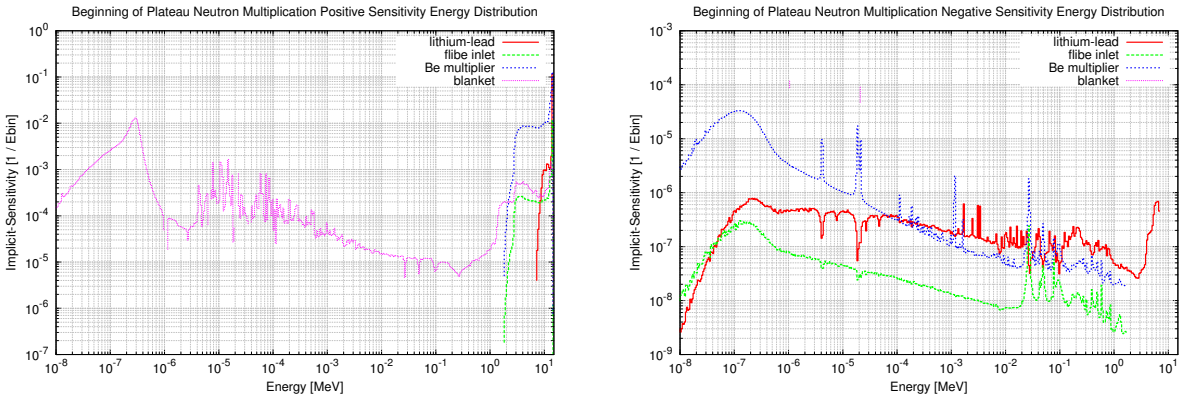

Figure 4.24: (Left) positive and (negative) sensitivity energy distributions for M_n at BOP versus spatial region.

resonance energies upon $\mathrm{Pu(n, fission)}$ reactions. The beryllium multiplier provides its multiplication at high energies, where the $^{9}\mathrm{Be(n, 2n)}$ reaction is present. The lithium-lead and

flibe inlet regions provide the slightest bit of multiplication at energies above (n, 2n) thresholds. The beryllium multiplier, lithium-lead, and flibe inlet regions are strong absorbers at resonance and thermal energies. This is where neutrons are sacrificed for the breeding of tritium.

The MPU CR sensitivity energy distributions are shown in Figure 4.25, binned versus spatial region. This blanket destroys fissile fuel at thermal energies, does a mix of fuel

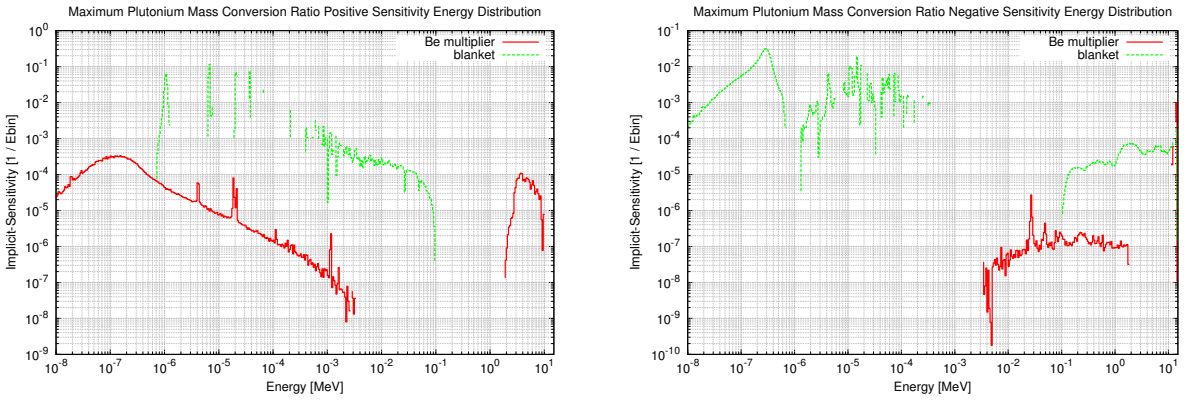

Figure 4.25: (Left) positive and (negative) sensitivity energy distributions for CR at MPU versus spatial region.

breeding and destruction within lower resonance regions, breeds within the upper resonance region, and destroys fuel again within the unresolved resonance and threshold energies. The beryllium multiplier region steals neutrons from potential fissions at thermal energies, steals neutrons that would thermalize somewhat and potentially breed in the blanket within upper resonance energies, steals neutrons from the unresolved resonance region that might fast-fission in the blanket, and provides a little bit of excess neutron production (and energy reduction) at threshold energies to enhance fast-fission.

4.5 Validation of adjoint-based methods

Two tests are performed in order to validated the sensitivity estimates of this work. In the first test, the number density of ^{6}Li within the lithium-lead region of BOL is increased 5% from $2.992 \times 10^{-4} \rightarrow 3.142 \times 10^{-4} \left[\frac{\text{atoms}}{\text{barn}-\text{cm}}\right]$. With the linear sensitivity coefficient of the TBR response to this input estimated to be +28.5%, the TBR is predicted to increase by +1.43%. For the second test, the number density of ^{56}Fe is decreased by 20% within all structural regions of BOL: the 1st wall, 2nd wall, 3rd wall, 4th wall, 5th wall, and final wall and decreasing their mass densities by almost the same amount. The linear sensitivity coefficients of TBR to these inputs are −0.0375%, −0.0820%, −0.136%, −0.413%, −0.0242%,

-0.000982%, respectively, for a total linear sensitivity of -0.693% and a predicted change in TBR of $+0.139\%$.

The unperturbed BOL TBR is calculated as 1.04071 ± 0.00009 (counting uncertainty). The TBR resulting from the ^{6}Li perturbation is 1.05450 ± 0.00009. This represents an increase of 1.33%, which is relatively 7.6% lower than the predicted increase of 1.43%: the correct direction and approximately the correct magnitude. The TBR resulting from the ^{56}Fe perturbation is 1.04415 ± 0.00009. This represents an increase of 0.331%, which is relatively 58% higher than the predicted increase of 0.139%: the correct direction, but quite a different magnitude.

While the first test is considered to somewhat validate the adjoint-based sensitivities, the second test is considered less successful in this respect. The 20% change in the ^{56}Fe density was necessary to bring a response change large enough to escape counting uncertainty noise. A 20% change in any input is a very large perturbation that likely lays outside the realm of linear sensitivity coefficient accuracy. Additionally, when the gross errors in resonant capture in ^{56}Fe within ENJEFF (see Section 3.8.4.3) are considered such a large discrepancy is almost to be expected.

4.6 Uncertainty analysis

In this section, estimates are made for the uncertainty of responses from three sources: (1) uncertain nuclear data; (2) statistical counting uncertainty of response estimates; and (3) statistical counting uncertainty of sensitivity estimates, which propagate to nuclear data uncertainty estimates–variances of variances. No restrictions were made in the uncertainty analysis. Every isotope present and each of their nuclear reactions are considered as sources of nuclear data uncertainty. Only at the very end, when uncertainty estimates are made, are the unimportant sources thrown to the side–no isotopes were hand-picked and no nuclear reactions were 'known' to be important a priori.

Uncertainties can be communicated in a number of ways, for example as an absolute uncertainty: 16 ± 2 [tangelos], a relative uncertainty: 16 [tangelos] $\pm$ 12.5%, or a confidence interval (the expected value minus and plus 2.58 standard deviations): "there is a 99% chance that the number of tangelos lays between 10.8 and 21.2". The uncertainty that an input contributes to a response can be reported and the intrinsic uncertainty of that input can be reported, irrespective of how sensitive that response is to the input. One can calculate the effective intrinsic uncertainty of an input p ($\bar{U_p}$) by dividing the overall relative uncertainty that an input contributes to a result R ($U_{R,p}$) by the overall sensitivity that result has to the input ($S_{R,p}$):

$$\bar{U_p} = \frac{U_{R,p}}{S_{R,p}}, \tag{4.1}$$

for example, an input that contributes 0.5% uncertainty to a response and 10% of the

sensitivity to that response has, itself, an effective 5% uncertainty. This single number represents how uncertain an input is for the design and neutron spectrum the uncertainty analysis was performed for. More generally, nuclear data uncertainties vary with neutron kinetic energy (see Section 4.6.3).

4.6.1 Nuclear response uncertainty estimates

The uncertainty tables in this section contain seven columns. The first two identify the nuclear reaction input and the third and fourth report the reaction rate (per source neutron) and sensitivity of the response (detailed in the caption) to that input. The fifth and sixth columns report the relative uncertainty from nuclear data uncertainties ('XS') and statistical counting uncertainties ('MC'), respectively with the units of pcm. One pcm is 10^{-5} or $\frac{1\%}{1000}$ or one 'per-centi-milli.' The seventh column shows the relative counting uncertainty of the nuclear data uncertainty estimate ('MCXS'), also in pcm. The cumulative row at the bottom of each table contains the estimate of the response, plus and minus its overall uncertainty (from the three sources) and the relative uncertainty that each source contributes individually, again in pcm.

Table 4.1 summarizes the most important contributors to BOL M_{th} uncertainty. The

Isotope	**Reaction**	**Rate [rxn/s]**	**Sensitivity**	**XS [pcm]**	**MC [pcm]**	**MC XS [pcm]**
U-235	(n, fission)	0.02434	40.19%	609.29	20.09	0.19
Li-6	(n, T)	0.95380	37.52%	74.88	8.74	0.01
Fe-56	(n, γ)	0.20378	13.53%	652.65	3.27	2.66
Be-9	(n, 2n)	0.60390	-7.50%	123.42	2.14	9.03
U-238	(n, γ)	0.20405	6.48%	82.56	3.89	0.24
Pb-208	(n, 2n)	0.09014	-6.22%	345.21	1.24	180.95
U-238	(n, fission)	0.00225	3.72%	24.53	3.35	0.05
Pb-206	(n, 2n)	0.04078	-3.09%	188.91	0.62	98.61
Li-7	(n, T)	0.07315	-3.00%	274.04	0.82	81.70
W-182	(n, γ)	0.05287	2.88%	252.11	2.20	8.31
Pb-207	(n, 2n)	0.03820	-2.42%	132.26	0.48	74.07
W-186	(n, γ)	0.04766	2.27%	66.81	1.26	1.06
Be-9	(n, γ)	0.03861	2.25%	110.32	0.65	0.53
Fe-56	(n, 2n)	0.01922	-2.02%	349.96	0.29	295.43
W-183	(n, γ)	0.03007	1.99%	158.31	1.81	3.13
...	...	...	...	...	...	...
Cumulative		0.77280 ± 0.00957		1168.15	23.15	410.37

Table 4.1: Top uncertainties of M_{th} at BOL.

^{6}Li(n, T), ^{9}Be(n, 2n), ^{238}U(n, γ), and ^{56}Fe(n, γ) reactions occur the most frequently, but the

^{235}U(n, fission) reaction contributes the most to sensitivity. The largest contributors to nuclear data uncertainty, however, are ^{56}Fe(n, γ), ^{235}U(n, fission), ^{56}Fe(n, 2n), and ^{208}Pb(n, 2n), with effective uncertainties of 4.8%, 1.5%, 17%, and 5.6%. Of all reactions that contribute more than 0.1% of absolute sensitivity, ^{238}U(n, 3n), ^{7}Li(n, 2n), and ^{19}F(n, T) have the highest effective intrinsic uncertainties, with 38%, 34%, and 29% respectively. Statistical counting uncertainty is well bounded by nuclear data uncertainty, but the counting uncertainty of the nuclear data uncertainty is somewhat high. The uncertainty of M_{th} from all sources sums to 1.2% at BOL, for a 99% confidence interval of $0.75 - 0.78$.

Table 4.2 contains the top uncertainties for TBR at BOP. The most frequent reactions are

Isotope	**Reaction**	**Rate [rxn/s]**	**Sensitivity**	**XS [pcm]**	**MC [pcm]**	**MC XS [pcm]**
Li-6	(n, T)	0.97393	86.75%	173.05	20.50	0.02
Li-7	(n, T)	0.07509	6.51%	584.37	1.74	133.82
Be-9	(n, 2n)	0.61647	1.82%	33.65	0.52	10.92
Be-9	(n, T)	0.01041	0.89%	83.54	0.34	21.82
Fe-56	(n, γ)	0.17800	-0.87%	41.99	0.20	2.52
U-238	(n, γ)	0.35971	-0.78%	10.11	0.47	0.10
Pu-239	(n, fission)	0.17783	-0.75%	5.92	0.45	0.04
Pu-239	(n, γ)	0.10703	-0.53%	6.49	0.32	0.07
Pu-240	(n, γ)	0.09078	-0.35%	7.19	0.28	0.45
F-19	(n, T)	0.00332	0.29%	83.30	0.07	87.68
Be-9	(n, γ)	0.03563	-0.24%	11.70	0.07	0.84
Li-7	(n, γ)	0.03205	-0.20%	29.92	0.05	5.80
W-182	(n, γ)	0.05425	-0.18%	15.35	0.13	5.58
W-186	(n, γ)	0.04788	-0.17%	5.13	0.10	1.37
Pu-241	(n, fission)	0.04137	-0.17%	1.93	0.10	0.03
...	...	...	...	...	...	...
Cumulative		1.06273 ± 0.03569		624.42	20.60	3299.23

Table 4.2: Top uncertainties of TBR at BOP.

^{6}Li(n, T), ^{9}Be(n, 2n), ^{238}U(n, γ), ^{56}Fe(n, γ), ^{239}Pu(n, fission), and ^{239}Pu(n, γ) and those that the response is most sensitive to are ^{6}Li(n, T), ^{7}Li(n, T), ^{9}Be(n, 2n), ^{9}Be(n, T), and ^{56}Fe(n, γ). The largest contributors to nuclear data uncertainty are ^{7}Li(n, T), ^{6}Li(n, T), ^{9}Be(n, T), and ^{19}F(n, T), for effective intrinsic uncertainties of 9.0%, 0.20%, 9.4%, and 29%, respectively. The reactions that contribute more than 0.1% of absolute sensitivity with the highest effective intrinsic uncertainty are ^{19}F(n, T), ^{7}Li(n, γ), ^{9}Be(n, T), ^{7}Li(n, T), and ^{182}W(n, 102), with effective intrinsic uncertainties of 29%, 15%, 9.4%, 9.0%, and 8.6%. Statistical counting uncertainty is once again well bounded by nuclear data uncertainty, but the counting uncertainty of nuclear data uncertainty is 5× larger. This means that while the order of magnitude of the uncertainty estimate is likely correct, very few significant figures should be trusted. The overall uncertainty of TBR from all sources (but mostly from counting

uncertainty of the nuclear data uncertainty) is 3.4% at BOP, for a 99% confidence interval of $1.0 - 1.2$.

The top uncertainties for M_n at MPU are shown in Table 4.3 below. The reactions

Isotope	**Reaction**	**Rate [rxn/s]**	**Sensitivity**	**XS [pcm]**	**MC [pcm]**	**MC XS [pcm]**
Be-9	(n, 2n)	0.61871	51.49%	866.09	14.61	2.69
Pu-239	(n, fission)	0.15420	19.52%	134.70	11.71	0.06
Pu-241	(n, fission)	0.06383	8.09%	251.30	4.85	0.95
Pb-208	(n, 2n)	0.09012	6.79%	375.57	1.36	36.77
Pb-206	(n, 2n)	0.04077	3.07%	186.96	0.61	20.08
Pb-207	(n, 2n)	0.03819	2.88%	156.45	0.58	15.09
Fe-56	(n, 2n)	0.01922	1.47%	255.99	0.21	71.09
Cm-245	(n, fission)	0.00871	1.10%	63.33	0.66	0.08
F-19	(n, 2n)	0.00940	0.75%	75.93	0.18	4.67
Li-7	(n, 2n)	0.00502	0.40%	140.80	0.10	32.42
U-238	(n, fission)	0.00296	0.38%	2.33	0.26	0.00
Li-7	(n, T)	0.07472	-0.27%	29.56	0.07	14.60
Pb-204	(n, 2n)	0.00233	0.18%	12.23	0.04	1.51
Cr-52	(n, 2n)	0.00180	0.14%	28.48	0.02	17.95
Am-242	(n, fission)	0.00108	0.14%	9.73	0.10	0.03
...	...	...	...	...	...	...
Cumulative		1.18160 ± 0.01261		1062.69	19.43	93.33

Table 4.3: Top uncertainties of M_n at MPU.

that the response is most sensitive to are among the most frequent reactions: ^{9}Be(n, 2n), ^{239}Pu(n, fission), ^{241}Pu(n, fission), ^{208}Pb(n, 2n), and ^{206}Pb(n, 2n). The largest contributors to nuclear data uncertainty are the ^{9}Be(n, 2n), ^{208}Pb(n, 2n), ^{56}Fe(n, γ), and ^{241}Pu(n, fission) reactions, which have effective uncertainties of 1.7%, 5.5%, 17%, and 3.1%, respectively. The reactions with non-negligible sensitivity and the highest effective intrinsic uncertainty are ^{7}Li(n, 2n), ^{52}Cr(n, 2n), ^{56}Fe(n, 2n), and ^{19}F(n, 2n), with values of 35%, 21%, 17%, and 10%, respectively. Both flavors of counting uncertainty are well bounded by nuclear data uncertainty for this response, so the overall uncertainty estimate of 1.1% and estimated 99% confidence interval of $1.15 - 1.21$ can be trusted reasonably well.

Table 4.4 contains the top uncertainties for CR at EOP. The ^{238}U(n, γ), ^{239}Pu(n, fission), ^{240}Pu(n, γ), ^{239}Pu(n, γ), and ^{240}Pu(n, fission) reactions contribute the most to sensitivity. The ^{238}U(n, γ), ^{240}Pu(n, γ), ^{244}Cm(n, γ), ^{241}Pu(n, fission), ^{239}Pu(n, fission), ^{239}Pu(n, γ), and ^{245}Cm(n, fission) reactions contribute the most to nuclear data uncertainty, with effective intrinsic uncertainties of 1.3%, 2.1%, 4.4%, 1.5%, and 0.76%, respectively. The reactions that possess the largest effective intrinsic an non-negligible sensitivity are ^{247}Cm(n, fission), ^{155}Eu(n, γ), ^{7}Li(n, T), ^{243}Am(n, γ), and ^{246}Cm(n, γ), with values of 16%, 10%, 7.8%, 6.3%, 5.8%, respectively. Counting uncertainty is well bounded by nuclear data uncertainty, but the

Isotope	**Reaction**	**Rate [rxn/s]**	**Sensitivity**	**XS [pcm]**	**MC [pcm]**	**MC XS [pcm]**
U-238	(n, γ)	0.17512	58.18%	740.27	40.73	1.72
Pu-239	(n, fission)	0.12752	-37.17%	282.49	18.59	0.14
Pu-240	(n, γ)	0.08519	28.60%	586.02	22.88	3.99
Pu-239	(n, γ)	0.07664	-22.51%	261.37	11.26	0.24
Pu-241	(n, fission)	0.06962	-20.44%	298.78	10.22	0.53
Cm-244	(n, γ)	0.02348	7.83%	344.23	7.04	9.39
Pu-241	(n, γ)	0.02577	-7.60%	96.67	3.80	0.10
Cm-245	(n, fission)	0.02262	-6.65%	218.28	3.33	0.16
Pu-238	(n, γ)	0.00627	2.17%	55.25	1.09	0.21
Am-241	(n, γ)	0.00387	-1.14%	34.07	0.57	0.07
Cm-245	(n, γ)	0.00338	-1.00%	39.86	0.50	0.05
Li-6	(n, T)	0.64414	0.86%	1.72	0.23	0.02
Cm-246	(n, γ)	0.00147	0.49%	28.85	0.35	0.41
Fe-56	(n, γ)	0.25548	0.44%	22.77	0.09	6.65
Cm-247	(n, fission)	0.00094	-0.28%	45.44	0.17	0.79
...	...	...	...	...	...	...
Cumulative		0.88051 ± 0.03072		1146.68	53.25	3294.86

Table 4.4: Top uncertainties of CR at EOP.

counting uncertainty of nuclear data uncertainty is 3× larger. This means that perhaps only one significant figure of the uncertainty estimate can be trusted: 3.5% for a 99% confidence interval of $0.8 - 1.0$.

Top uncertainties for TBR at BOT are shown below in Table 4.5. Like with TBR at BOP, the most sensitive reactions are $^{6}\mathrm{Li}(\mathrm{n}, \mathrm{T})$, $^{7}\mathrm{Li}(\mathrm{n}, \mathrm{T})$, $^{9}\mathrm{Be}(\mathrm{n}, 2\mathrm{n})$, $^{9}\mathrm{Be}(\mathrm{n}, \mathrm{T})$, and $^{56}\mathrm{Fe}(\mathrm{n}, \gamma)$. Again, similarly to BOP, the largest contributors to nuclear data uncertainty are the $^{7}\mathrm{Li}(\mathrm{n}, \mathrm{T})$, $^{6}\mathrm{Li}(\mathrm{n}, \mathrm{T})$, $^{9}\mathrm{Be}(\mathrm{n}, \mathrm{T})$, and $^{19}\mathrm{F}(\mathrm{n}, \mathrm{T})$, with effective intrinsic uncertainties of 9.1%, 0.20%, 9.4%, and 29%, respectively. The non-negligible reactions with the highest effective intrinsic uncertainties are identical to BOP, except that the $^{182}\mathrm{W}(\mathrm{n}, \gamma)$ reaction is 8.8% instead of 8.6%.

Overall, the estimated nuclear data uncertainties of important figures of merit in LIFE were quite moderate. The uncertainty in TBR remained below 1% for all time steps, staying around 0.6% for all time-steps except during EOP, when it was 0.9%. The CR uncertainty was highest at BOL at 1.8% and fell to around 1.2% thereafter. The uncertainty for M_{th} started at 1.2%, fell to around 0.6% at BOP, jumped up to 0.8% at MPU, and then fell back down to 0.6% for EOP and BOT. The M_n uncertainty started at 1.4%, fell to 1% during the power plateau, and then rose to 1.2% during BOT.

Isotope	**Reaction**	**Rate [rxn/s]**	**Sensitivity**	**XS [pcm]**	**MC [pcm]**	**MC XS [pcm]**
Li-6	(n, T)	0.95265	86.96%	173.35	20.56	0.02
Li-7	(n, T)	0.07401	6.58%	596.17	1.78	113.77
Be-9	(n, 2n)	0.61205	1.79%	32.71	0.51	9.02
Be-9	(n, T)	0.01041	0.92%	85.72	0.35	18.18
Fe-56	(n, γ)	0.14276	-0.68%	32.37	0.18	1.53
F-19	(n, T)	0.00332	0.29%	85.40	0.08	74.50
Pu-239	(n, fission)	0.07561	-0.24%	1.89	0.14	0.01
U-238	(n, γ)	0.12806	-0.22%	2.77	0.15	0.04
Be-9	(n, γ)	0.02950	-0.19%	9.49	0.06	0.56
Pu-239	(n, γ)	0.04595	-0.18%	2.23	0.11	0.02
Pu-240	(n, γ)	0.06142	-0.17%	3.51	0.16	0.16
W-182	(n, γ)	0.04907	-0.16%	13.71	0.12	4.38
W-186	(n, γ)	0.04281	-0.15%	4.62	0.09	1.08
Li-7	(n, γ)	0.02422	-0.15%	21.87	0.04	3.94
Pu-241	(n, fission)	0.04114	-0.12%	1.24	0.07	0.02
...	...	...	...	...	...	...
Cumulative		1.04037 $\pm$ 0.19946		634.97	20.65	19161.73

Table 4.5: Top uncertainties of TBR at BOT.

4.6.2 Comparison of implicit and explicit uncertainty estimates

In Table 4.6, the explicit uncertainties of TBR at BOP are shown. This table, which has

Isotope	**Reaction**	**Rate [rxn/s]**	**Sensitivity**	**XS [pcm]**	**MC [pcm]**	**MC XS [pcm]**
Li-6	(n, T)	0.97393	91.64%	182.81	21.66	0.01
Li-7	(n, T)	0.07509	7.07%	634.18	1.89	24.36
Be-9	(n, T)	0.01041	0.98%	92.07	0.37	3.29
F-19	(n, T)	0.00332	0.31%	91.09	0.08	10.66
Cumulative		1.06273 $\pm$ 0.00716		672.59	21.75	26.79

Table 4.6: Explicit uncertainties of TBR at BOP.

only four entries, should be viewed in the context of its implicit sibling Table 4.2. Explicit reaction rates and sensitivities are trivially related, with the latter equaling the individual contributions to the sum of the former. ^{7}Li(n, T) and ^{6}Li(n, T) contribute the most to uncertainty, with effective intrinsic uncertainties of 9.0% and 0.20%, each. The ^{9}Be(n, T) and ^{19}F(n, T) intrinsic uncertainties remain essentially the same, but with TBR more explicitly sensitive to the reactions, the overall uncertainty estimate increases. These four isotopes were the main players in the implicit uncertainty analysis. The nuclear data overwhelms

both varieties of counting uncertainty, so much trust is placed on the explicit nuclear data uncertainty estimate of 0.67%. This is 6× lower than the implicit sensitivity estimate. Consequently, the explicit 99% confidence interval is much tighter: $1.04 - 1.08$.

4.6.3 Intrinsic uncertainty of nuclear data

There are a few ways to represent the intrinsic uncertainty of nuclear data. The most intuitive way, shown below in Figure 4.26, displays the expected value of the $^{6}\mathrm{Li}(\mathrm{n}, \mathrm{T})$ microscopic cross-section in black and its range of possible values (plus and minus one standard deviation) in red. This relatively simple cross-section is the most important for TBR in the design (see

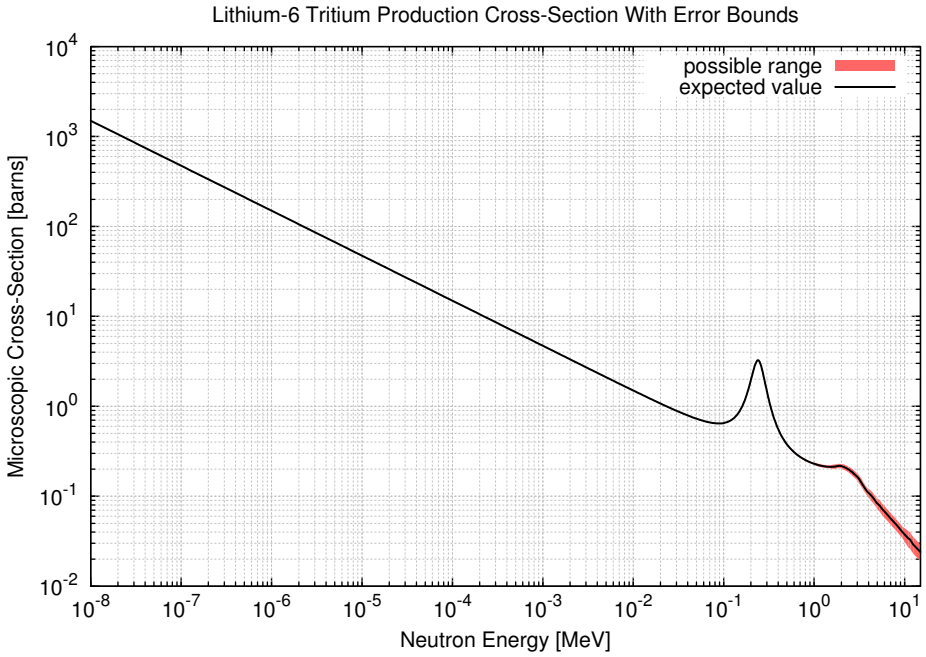

Figure 4.26: The $^{6}\mathrm{Li}$ tritium production cross-section has low uncertainty at thermal energies, but not insignificant uncertainty at high energies. Nominal nuclear data is courtesy of NADS [McKinley *et al.*, 2004].

Tables 4.2 and 4.5). Since sensitivity to this cross-section is largest at thermal energies, where the uncertainty is quite low, the effective intrinsic uncertainty is also low.

The intrinsic uncertainty of nuclear data can also be shown as relative standard deviations and correlation matrices, like in Figure 4.27 for $^{9}\mathrm{Be}(\mathrm{n}, 2\mathrm{n})$. This threshold reaction is the most important for M_n in the design (see Table 4.3). Relative uncertainty is large at the threshold and then $2 - 3\%$ above the threshold.

The covariance matrix for $^{56}\mathrm{Fe}(\mathrm{n}, \gamma)$ is depicted below in Figure 4.28. This reaction is

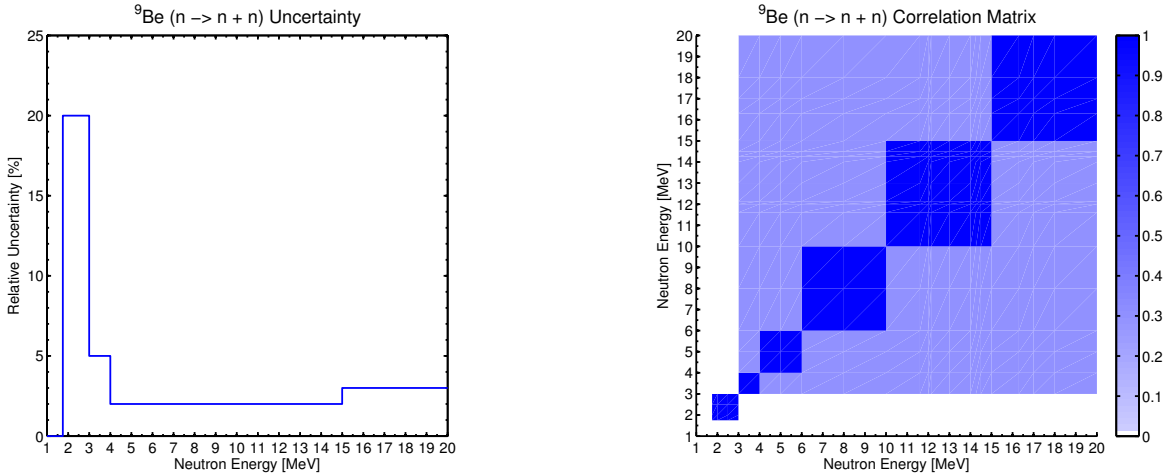

Figure 4.27: The (left) relative uncertainty and (right) correlation matrix for $^{9}\mathrm{Be}(\mathrm{n}, 2\mathrm{n})$. Covariances courtesy of [Little *et al.*, 2008; DOE/NNSA, 2011].

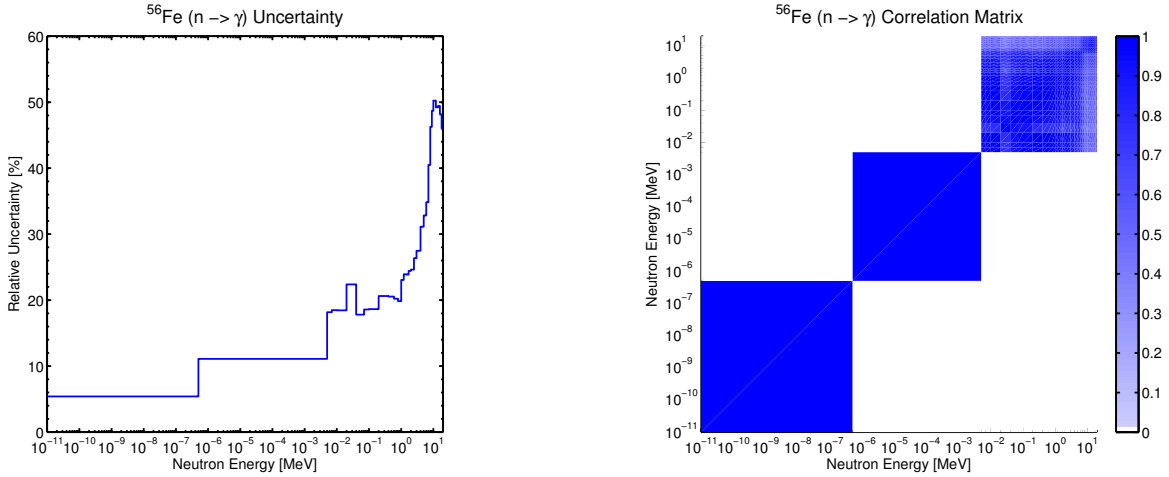

Figure 4.28: The (left) relative uncertainty and (right) correlation matrix for $^{26}\mathrm{Fe}(\mathrm{n}, \gamma)$. Covariances courtesy of [Little *et al.*, 2008; DOE/NNSA, 2011].

responsible for most of the parasitic absorption in the system and subsequently, most figures of merit are negatively influenced by it (in a performance sense). At thermal and resonance energies, the uncertainty is estimated to be around 6% and around 12%. In the unresolved region, relative uncertainties climb to around 20% and then $30-50\%$ in the threshold region.

Figure 4.29 shows the covariances for the $^{235}\mathrm{U}(\mathrm{n}, \mathrm{fission})$ reaction. This reaction is responsible for a large amount of the thermal multiplication at BOL (see Table 4.1). Its overall uncertainty lays roughly between $1-2\%$ for the entire energy spectrum except between

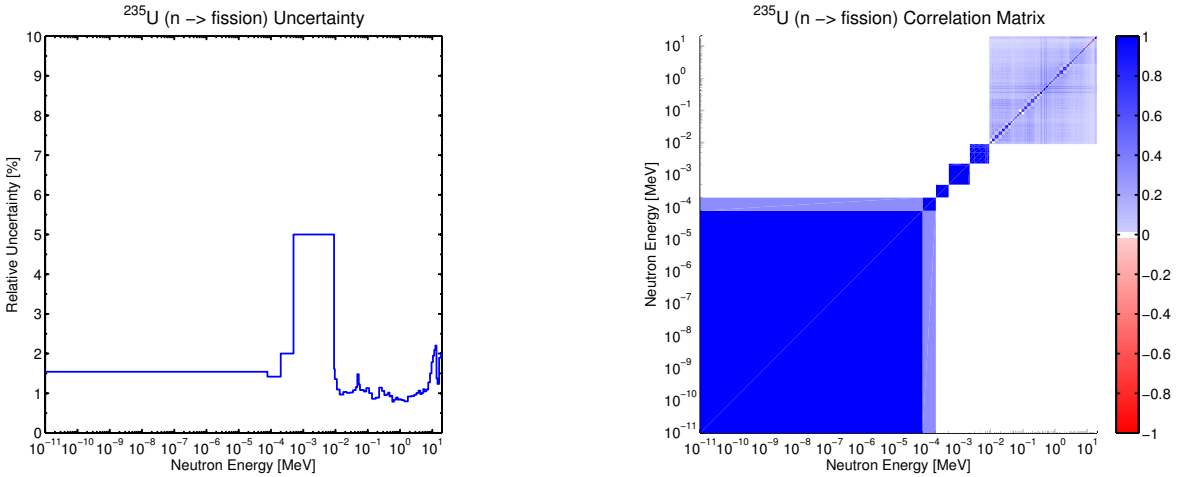

Figure 4.29: The (left) relative uncertainty and (right) correlation matrix for $^{2}\mathrm{U}(\mathrm{n},3)5$fission. Covariances courtesy of [Shibata *et al.*, 2011; JAEA/NDC, 2011b].

$0.5 - 10\,[\mathrm{keV}]$, where it jumps to 5%.

The $^{238}\mathrm{U}(\mathrm{n},\gamma)$ reaction's covariances are shown below in Figure 4.30. This resonance

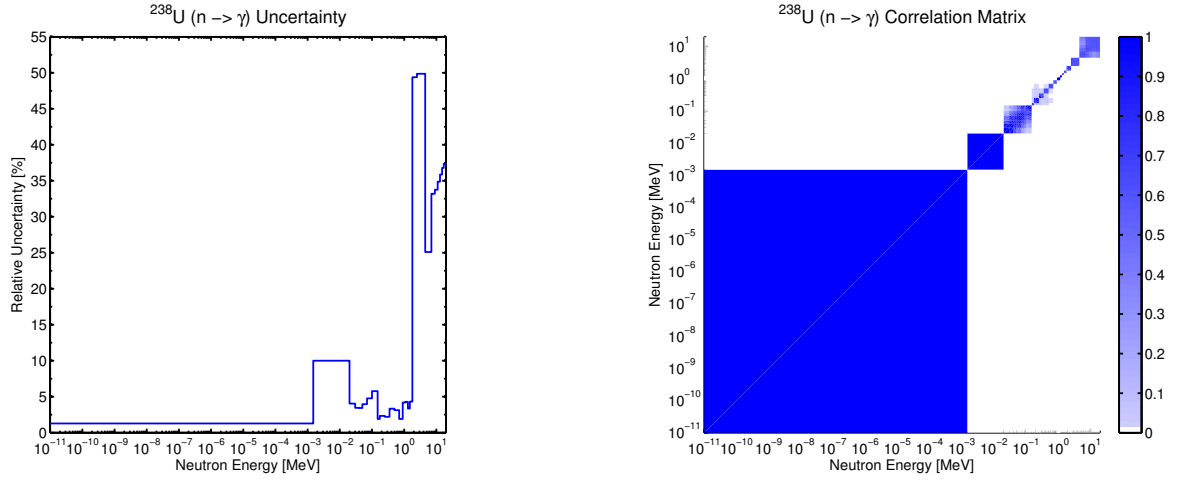

Figure 4.30: The (left) relative uncertainty and (right) correlation matrix for $^{238}\mathrm{U}(\mathrm{n},\gamma)$. Covariances courtesy of [Shibata *et al.*, 2011; JAEA/NDC, 2011b].

reaction is responsible for most of the fertile breeding during the lifetime of the design (see Table 4.4). The uncertainties within the range where sensitivities are highest are quite small at around 1.3%. In the unresolved energy range, it jumps to 10%, then back below 5%. At threshold energies, the uncertainty is between $25 - 50\%$, and lowest near the fusion neutron

energy of 14.08 [MeV].

In Figure 4.31 below, effective intrinsic nuclear data uncertainties are plotted versus explicit sensitivity for all 2328 explicit sensitivities (binned by isotopic reaction type) generated in this work. The overall negative correlation between the two implies that the less certain

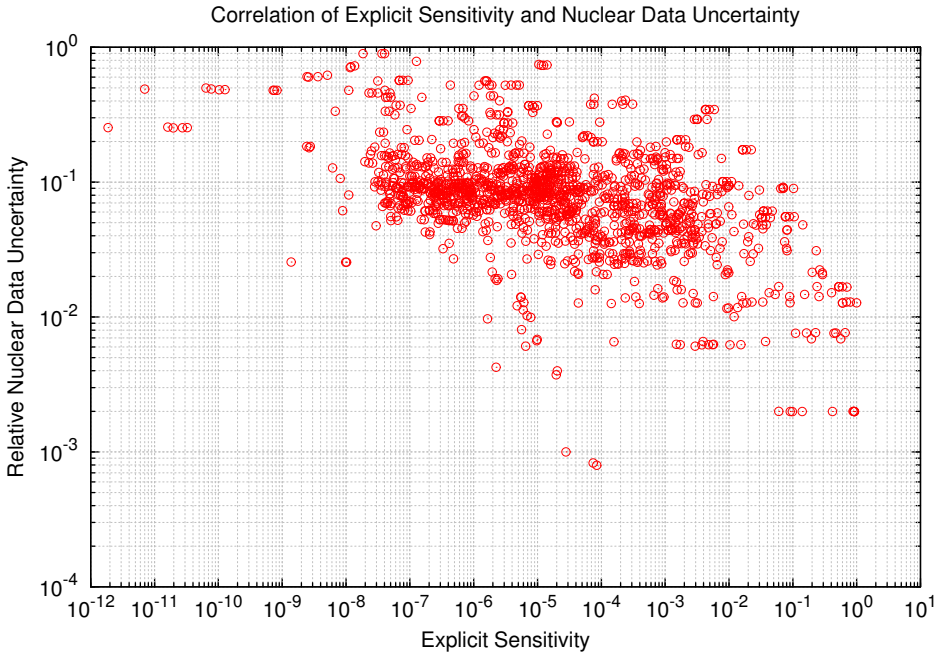

Figure 4.31: Effective intrinsic nuclear data uncertainty is negatively correlated with explicit sensitivity, with a correlation coefficient of -0.071 .

a nuclear datum is, the less explicitly sensitive the design is to it. It is expected that this correlation coefficient is more negative for an LWR.

4.6.4 Comparison of nuclear data covariances and variances

In the occasional uncertainty analysis, a comparison is done between using full covariance matrices and using just the diagonal variances [Downar, 1991; Taiwo *et al.*, 2005]. This work follows their example by doing the same analysis, effectively changing Equation 2.1 from a matrix product to an element-wise product. By default, the full covariance matrices are used, but in order to determine how important off-diagonal terms are, a switch was programmed to allow for a comparison.

Table 4.7 shows the diagonal-only uncertainties of M_{th} at BOL. This table should be studied alongside the 'full-covariance' version in Table 4.1. The overall nuclear data uncertainty is reduced. The $^{56}\mathrm{Fe}(\mathrm{n},16)$, $^{208}\mathrm{Pb}(\mathrm{n},16)$, $^{7}\mathrm{Li}(\mathrm{n},205)$, $^{9}\mathrm{Be}(\mathrm{n},16)$, $^{206}\mathrm{Pb}(\mathrm{n},16)$,

Isotope	**Reaction**	**Rate [rxn/s]**	**Sensitivity**	**XS [pcm]**	**MC [pcm]**	**MC XS [pcm]**
U-235	(n, fission)	0.02434	40.20%	608.43	20.10	0.19
Li-6	(n, T)	0.95380	37.52%	74.55	8.74	0.01
Fe-56	(n, γ)	0.20378	13.53%	652.16	3.27	2.66
Be-9	(n, 2n)	0.60390	-7.39%	57.49	2.11	6.05
U-238	(n, γ)	0.20405	6.48%	82.55	3.89	0.24
Pb-208	(n, 2n)	0.09014	-6.20%	254.70	1.24	158.98
U-238	(n, fission)	0.00225	3.72%	12.73	3.35	0.05
Pb-206	(n, 2n)	0.04078	-3.08%	139.87	0.62	85.46
Li-7	(n, T)	0.07315	-3.00%	196.48	0.82	69.49
W-182	(n, γ)	0.05287	2.88%	252.03	2.20	8.31
Pb-207	(n, 2n)	0.03820	-2.41%	97.52	0.48	64.56
W-186	(n, γ)	0.04766	2.27%	66.79	1.26	1.06
Be-9	(n, γ)	0.03861	2.25%	109.32	0.65	0.53
Fe-56	(n, 2n)	0.01922	-2.01%	258.26	0.29	259.43
W-183	(n, γ)	0.03007	1.99%	157.32	1.81	3.13
...	...	...	...	...	...	...
Cumulative		0.77280 $\pm$ 0.00879		1076.54	23.15	366.24

Table 4.7: Top 'diagonal-only' uncertainties of M_{th} at BOL.

^{9}Be(n, 205), ^{207}Pb(n, 16), ^{19}F(n, 16), and ^{19}F(n, 205) reactions all contribute 10's of pcm less to nuclear data uncertainty. The effective intrinsic uncertainties are all reduced–on average by a few percent. In general, the off-diagonal terms in covariance matrices increase uncertainty estimates–by an amount that is not insignificant–and should not be left out in an uncertainty analysis.

4.6.5 Uncertainty associated with an imperfect uncertainty analysis

In order to neglect one uncertainty among others, it should be at least 1 or 2 orders of magnitude smaller than the largest uncertainty. Otherwise, the total uncertainty will be erroneous by more than a percent or so (see Figure 4.32). In practice, one cannot argue that an uncertainty or error is bounded by another until they are both quantified. Of course, once an uncertainty is quantified, it might as well be propagated.

Explicit sensitivities and explicit results uncertainties all have small counting uncertainties that are well bounded by nuclear data uncertainties. This is straightforward to achieve since relative counting uncertainty and explicit sensitivity are negatively correlated–regions of phase space that a response is less explicitly sensitive to tend to be the regions that 'see' fewer particles. For this same reason, the counting uncertainty of nuclear data estimates are negatively correlated. Evidence for these trends are shown below in Figure 4.33. In these

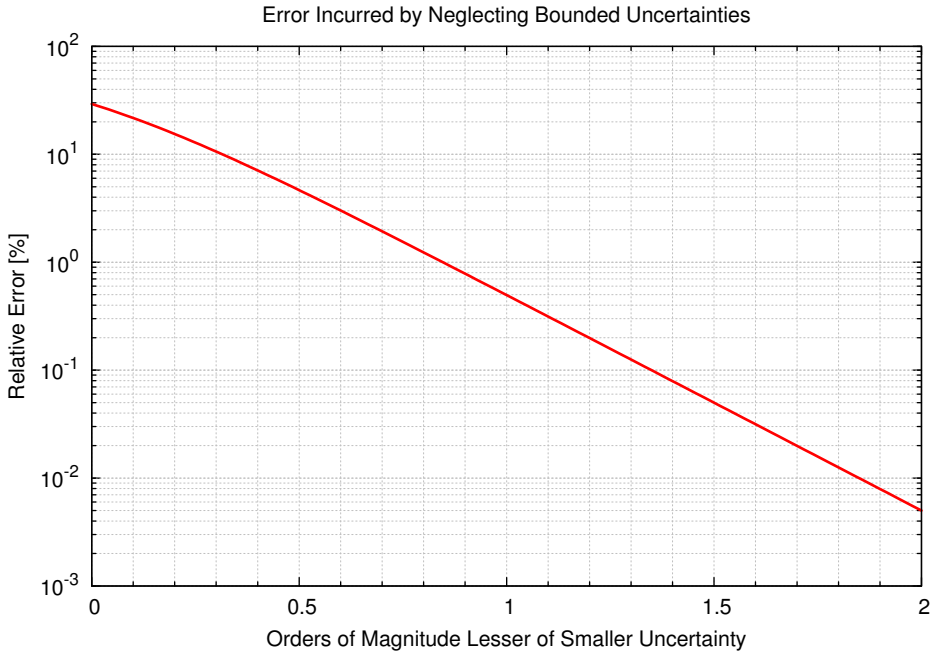

Figure 4.32: Penalty incurred by neglecting a smaller compounding uncertainty.

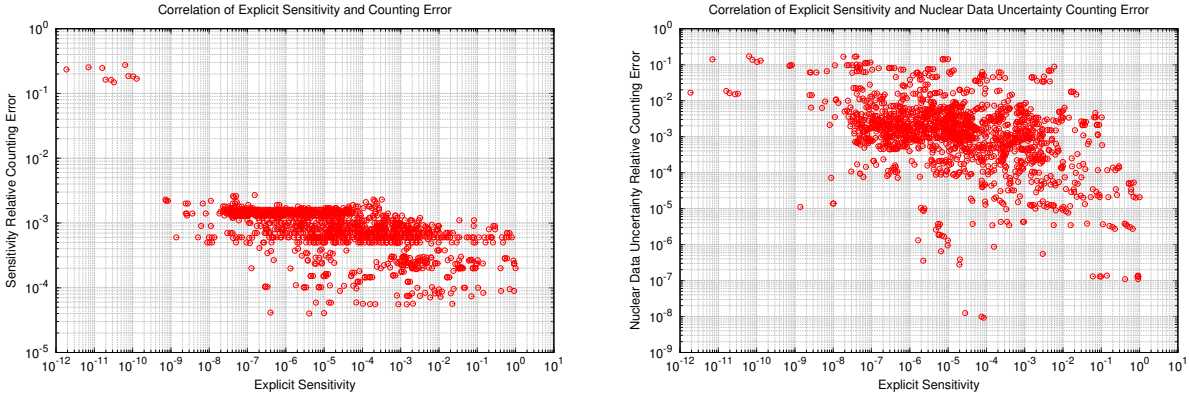

Figure 4.33: Counting uncertainty and counting uncertainty of nuclear data uncertainty are negatively correlated with explicit sensitivity, with correlation coefficients of -0.011 and -0.032, respectively.

graphs, both varieties of counting uncertainty are plotted against their associated explicit sensitivities for all 2328 explicit sensitivities that were calculated in this work. The inputs that contribute the highest relative counting uncertainty to a response are associated with the lowest sensitivity. Combining the two, absolute counting uncertainties remain small.

Errors associated with implicit sensitivity in this work are more complicated. For example, there are errors associated with multi-group cross-sections (see Section 3.8.4 and [Seifried *et al.*, 2011]). Several of the multi-group cross-sections are deemed inadequate for forward transport and therefore vicariously, so are some of the adjoint distributions. This is not addressed fully. Additionally, due to computational restraints, adjoint distributions contain a large degree of statistical counting uncertainty. This accounts for a large amount of the counting uncertainty of implicit sensitivity estimates. Another large contributor to counting uncertainty in implicit sensitivities is the reliance on (the sometimes incredibly inefficient) tally tagging for source distributions. Consider, for example elastic scattering in the tungsten armor as a simultaneous sink and source. Virtually every particle passes through this region (some multiple times) and provide some information to estimate the average particle track-length within the region and therefore the probability and rate of elastic scatters. The region is painted onto the first wall. It is razor thin. There are very few simulated scatters with which to construct a post-elastic-scatter distribution. Even the source rate uncertainty is high. Physically, the elastic scatter 'source rate' is identical to the elastic scatter 'sink rate,' but numerically this is rarely achieved, so elastic scatter sink and source rates end up unbalanced. These large errors result in very large implicit sensitivity estimates. This problem was so bad that elastic scatter sensitivities are suppressed when the imbalance is too large. The easiest solution to this is to sample more histories. A smarter solution, however, was not found.

4.7 Convergence study (or the lack thereof)

One important aspect of any computational analysis is a convergence study. In such a study, the fidelity of the simulation is increased (usually by shrinking discrete bins or increasing the Monte Carlo histories) until the precision of results cease to change. For Monte Carlo simulations, finer tally bins receive less information and produce per-bin results that are less certain. In order to preserve per-bin precision more histories must be run. Consequently, clock-time scales super-linearly (considering processor, I/O, and memory requirements) with Monte Carlo model fidelity when per-bin precisions are preserved. Convergence studies ensure that a simulation can produce results at no finer a precision, but can be cumbersome.

There was not enough time to perform a convergence study for this work. Instead, all simulations were run with the highest possible fidelity–given computing and time restrictions–in a series of 'hero' runs. It is the hope of the author that the simulations are of sufficient fidelity. Explicit and adjoint calculations were performed with 10^8 histories and implicit calculations used 10^7 histories. Propagation of counting uncertainties to all results was considered an acceptable replacement for a convergence study. 615 bins were used to discretize all quantities with respect to neutron kinetic energy. This degree of binning (50 equi-lethargic bins per decade) has been considered acceptable for other works [Cullen, 2005]. 16 bins were used

for directional binning. This number was, once again, not chosen from an explicit study of precision convergence, but rather was agreed to be acceptable. Spatial regions in the system were not subdivided. It was suggested that adjoint distributions, specifically their angular dependence would change versus location in the system. This deficiency and the others are left for future work.

Chapter 5

Conclusion

Overall, this effort–the quantification of uncertainties in neutron transport simulations using adjoint-based methods with MCNP6–was successful. Adjoint bilinear functionals were constructed with the highest available fidelity: three-dimensional neutron transport, with explicit geometrical double heterogeneity, and continuous energy nuclear data (multi-group for adjoint calculations). From these functionals, sensitivities of important results to a million inputs were generated with just a handful of calculations. These sensitivities offered rich landscapes of information from which to garner physical intuition of a design to guide design improvement or material composition optimization. The sensitivities also served as the vessels to propagate the uncertainties of the million inputs to resultant figures of merit.

The analysis wasn't without its limitations, however. The sensitivity and uncertainty analyses were instantaneous in nature. These quantities were unable to propagate with time, to transition from reaction rate uncertainties to mass uncertainties and back again. With a system such as the DU-hybrid LIFE blanket that is designed to operated long periods of time, time-dependent sensitivities and uncertainties will compound and interact. End-of-life metrics, like k_{eff}, the quantity of certain radioisotopes, and the material nature of fuel components can be important parts of a design effort. Only a time-dependent uncertainty analysis can address how these figures of merit are prone to change with uncertain simulation inputs.

The majority of effort in this work was in extracting information from MCNP6 simulations in the correct manner to construct adjoint bilinear functionals, since MCNP6 is not designed to do so. Angular distributions and source distributions were cumbersome (in terms of programming and simulation runtimes) and could be greatly improved. Perhaps a better approach to adjoint-based uncertainty quantification with MCNP6 would be one that embeds sensitivity estimation within the code its self instead constructing them externally, from disparate results. Fewer simplifications would be made since terms could be constructed on a per-particle basis, instead of the approximate per-tally-bin (space, energy, and direction) basis and correlations between quantities could be properly taken into account. Addition-

ally, continuous-energy adjoint transport would greatly improve uncertainty analyses, when and if the methods are developed. Inflexible multi-group cross-section libraries wouldn't be necessary and a single consistent data set could be used for all transport calculations, with no multi-group errors.

Overall, the estimated nuclear data uncertainties of important performance metrics (TBR, CR, M_{th}, and M_n) for the depleted-uranium LIFE blanket were quite moderate. There were small changes in the estimated uncertainty of each, but for the large part, that of TBR stayed below 1%, that of CR remained $1-2\%$, and those of M_{th} and M_n hovered around 1%. The counting uncertainties of all metrics remained in the tens of pcm range ($< 0.1\%$). The uncertainty due to counting uncertainty of nuclear data uncertainty varied widely. For the most part it was quite small (less than 1% or so), but the metrics at a few select time-steps were 8% and 20% due to an insufficient number of Monte Carlo histories for the adjoint transport calculations.

Statistical counting uncertainties were propagated for all physical quantities extracted and constructed from MCNP6. Some of those counting uncertainties were so large as to mask nuclear data uncertainties–namely for adjoint distributions and some source distributions, so additional computational time and some improvement to the tally tagging scheme are necessary to reduce these variances of variances in the future. It is important to recognize that these excessive uncertainties were born out of the very analysis whose purpose was to estimate uncertainties. No analysis, even an uncertainty analysis, is without uncertainty or bias. Counting uncertainties were addressed in this analysis, but errors brought from multi-group cross-sections, were not. Diffusion-based or two-dimensional analyses incorporate their own simplifications that are not easily quantified. It is difficult to say what ultimately has a greater effect on an analysis–the uncertainties that are quantified, or the uncertainties that are not.

Bibliography

RP Abbott, MA Gerhard, KJ Kramer, JF Latkowski, KL Morris, PF Peterson, and JE Seifried. Thermal and mechanical design aspects of the LIFE engine. *Fusion Science and Technology*, 56(2):618–624, August 2009. Proceedings of the Eighteenth Topical Meeting on the Technology of Fusion Energy (TOFE) (Part 2).

HS Abdel-Khalik, PJ Turinsky, MA Jesse, TE Stover, JR Elkins, and MM Iqbal. Uncertainty quantification, sensitivity analysis, and data assimilation for nuclear system simulation. *Nuclear Data Sheets*, 109(12), December 2008.

G Aliberti, G Palmiotti, M Salvatores, TK Kim, TA Taiwo, M Anitescu, I Kodeli, E Sartori, JC Bosq, and J Tommasi. Nuclear data sensitivity, uncertainty, and target accuracy assessment for future nuclear systems. *Annual of Nuclear Energy*, 33(8):700–733, 2006. `http://dx.doi.org/0.1016/j.anucene.2006.02.003`.

R Arcilla. ENDF/B-VII.0 in ACE-format. Technical Report RSICC Data Package: DLC-226, Brookhaven National Laboratory — National Nuclear Data Center: BNL/NNDC, Upton, NY, 2007.

GI Bell and S Glasstone. *Nuclear Reactor Theory*. Van Nostrand Reinhold Company, New York, NY, 1970.

Brookhaven National Laboratory — National Nuclear Data Center: BNL/NNDC. Evaluated nuclear data library: ENDF/B-VI.8. `http://www-nds.iaea.org/point2004/`, January 2011.

Brookhaven National Laboratory — National Nuclear Data Center: BNL/NNDC. Evaluated nuclear data library: ENDF/B-VII.0. `http://www-nds.iaea.org/point2009/pt2009.htm`, January 2011.

Brookhaven National Laboratory — National Nuclear Data Center: BNL/NNDC. Us library of evaluated nuclear data: ENDF/B-VII.1beta3. `https://ndclx4.bnl.gov/gf/project/endf/`, August 2011.

FB Brown. Monte carlo parameter studies & uncertainty analyses with mcnp5. Technical Report LA-UR-04-2506, LANL, Los Alamos, NM, May 2008.

O Cabellos. Processing of the JEFF-3.1 cross section library into a continuous energy monte carlo radiation transport and criticality data library. Technical Report NEA/NSC/DOC(2006)18, Organization for Economic Co-Operation and Development — Nuclear Energy Agency — Data Bank: OECD/NEA/DB, Issy-les-Moulineaux, France, 2006.

DG Cacuci. *Sensitivity and Uncertainty Analysis*. Chapman & Hall/CRC, Boca Raton, 2003.

MB Chadwick, P Obložinský, M Herman, NM Greene, RD McKnight, DL Smith, PG Young, RE MacFarlane, GM Hale, SC Frankle, AC Kahler, T Kawano, RC Little, DG Madland, P Moller, RD Mosteller, PR Page, P Talou, H Trellue, MC White, WB Wilson, R Arcilla, CL Dunford, SF Mughabghab, B Pritychenko, D Rochman, AA Sonzogni, CR Lubitz, TH Trumbull, JP Weinman, DA Brown, DE Cullen, DP Heinrichs, DP McNabb, H Derrien, ME Dunn, NM Larson, LC Leal, AD Carlson, RC Block, JB Briggs, ET Cheng, HC Huria, ML Zerkle, KS Kozier, A Courcelle, V Pronyaev, and SC van der Marck. ENDF/B-VII.0: Next generation evaluated nuclear data library for nuclear science and technology. *Nuclear Data Sheets*, 107(12):2931–3118, December 2006. http://dx.doi.org/10.1016/j.nds.2006.11.001.

China Institute of Atomic Energy — China Nuclear Data Center: CIAE/CNDC. Chinese evaluated nuclear data library: CENDL-3.1. http://www.nea.fr/dbforms/data/eva/evatapes/cendl_31/, January 2011.

RR Coveyou, VR Cain, and KJ Yost. Adjoint and importance in monte carlo application. Technical Report ORNL-4093, ORNL, Oak Ridge, TN, 1967.

CSEWG. ENDF-6 formats manual. Technical Report BNL-90365-2009, BNL, Upton, NY, 2009.

DE Cullen. Tart 2005 a coupled neutron-photon 3-d combinatorial geometry time dependent monte carlo transport code. Technical Report UCRL-SM-218009, LLNL, Livermore, CA, November 2005.

DE Cullen. Definition of k-eff. http://home.comcast.net/~redcullen1/Papers/K-eff/K-eff.pdf, May 2009.

Department of Energy — National Nuclear Security Administration: DOE/NNSA. Low-fidelity covariance project: Low-fi. http://www.nndc.bnl.gov/lowfi/, January 2011.

TJ Downar. Advanced depletion perturbation methods and waste transmutation in the ifr. Technical Report DOE/ER/12812–T1, Purdue University, West Lafayette, IN, December 1991.

JJ Duderstadt and LJ Hamilton. *Nuclear Reactor Analysis*. John Wiley & Sons, New York, NY, 1976.

M Fratoni, KJ Kramer, JF Latkowski, RP Abbott, JE Seifried, and JJ Powers. Attainable burnup in a LIFE engine loaded with depleted uranium. May 2010. Presented at 2010 PHYSOR.

N García-Herranz, O Cabellos, J Sanz, J Juan, and JC Kuijper. Propagation of statistical and nuclear data uncertainties in monte carlo burn-up calculations. *Annals of Nuclear Energy*, 35(4):714–730, April 2008. http://dx.doi.org/10.1016/j.anucene.2007.07.022.

E Greenspan. Developments in perturbation theory. *Advances in Nuclear Science and Technology*, 9:181–268, 1976.

E Greenspan. On the adjoint space in reactor theory. *Annals of Nuclear Energy*, 3(5-6):323–327, 1976. http://dx.doi.org/10.1016/0306-4549(76)90059-1.

E Greenspan. *Sensitivity Functions for Uncertainty Analysis*. Plenum Press, New York, NY, 1982.

RL Iman and WJ Conover. A distribution-free approach to inducing rank correlation among input variables. *Communications in Statistics*, 11(3):311–334, 1982.

Japan Atomic Energy Agency — Nuclear Data Center: JAEA/NDC. Japanese evaluated neutron data library: JENDL-3.3. http://wwwndc.jaea.go.jp/jendl/j33/j33.html, January 2011.

Japan Atomic Energy Agency — Nuclear Data Center: JAEA/NDC. Japanese evaluated neutron data library: JENDL-4.0. http://wwwndc.jaea.go.jp/jendl/j40/j40.html, January 2011.

MA Jessee, ML Williams, and MD Dehard. Development of generalized perturbation theory capability within the SCALE code package. *Proceedings of M&C 2009*, May 2009. Presented at the 2009 International Conference on Mathematics, Computational Methods, and Reactor Physics (M&C 2009).

BC Kiedrowski and FB Brown. Mcnp calculations of subcritical fixed source and fission multiplication factors. *ANS Transactions*, 102(1):503–505, June 2010. Presented at the 2010 American Nuclear Society (ANS) Annual Meeting.

BC Kiedrowski, FB Brown, and PPH Wilson. Adjoint-weighted tallied for k-eigenvalue calculations with continuous-energy monte carlo. *Nuclear Science and Engineering*, 168(3):226–241, July 2011.

GF Knoll. *Radiation detection and measurement*. John Wiley & Sons, New York, NY, 2000.

KJ Kramer, JF Latkowski, RP Abbott, JK Boyd, JJ Powers, and JE Seifried. Neutron transport and nuclear burnup analysis for the laser inertial confinement fusion-fission energy (LIFE) engine. *Fusion Science and Technology*, 56(2):625–631, August 2009. Proceedings of the Eighteenth Topical Meeting on the Technology of Fusion Energy (TOFE) (Part 2).

KJ Kramer, JF Latkowski, RP Abbott, M Fratoni, JJ Powers, JE Seifried, and JM Taylor. The laser inertial fusion engine as a weapons-grade plutonium fuel burner. *ANS Transactions*, 102(1):91–92, June 2010. Presented at the 2010 American Nuclear Society (ANS) Annual Meeting.

KJ Kramer, M Fratoni, JF Latkowski, RP Abbott, TM Anklam, EM Beckett, AJ Bayramian, JA DeMuth, RJ Deri, TD De La Rubia, AM Dunne, BS El-dasher, JC Farmer, A Lafuente, WR Meier, RW Moir, KL Morris, EI Moses, JJ Powers, S Reyes, RH Sawicki, JE Seifried, E Storm, and JM Taylor. Fusion-fission blanket options for the LIFE engine. *Fusion Science and Technology*, 60(1):72–77, August 2011. Proceedings of the Nineteenth Topical Meeting on the Technology of Fusion Energy (TOFE) (Part 1).

KJ Kramer. *Laser Inertial Fusion-based Energy: Neutronic Design Aspects of a Hybrid Fusion-Fission Nuclear Energy System*. PhD thesis, University of California, Berkeley, Berkeley, California, USA, May 2010.

Los Alamos National Laboratory — T-2 Nuclear Information Service: LANL/T-2. NJOY 99 nuclear data processing system. `http://t2.lanl.gov/codes/njoy99/`, July 2011.

J Lewins. *Importance — The Adjoint Function — The Physical Basis of Variational and Perturbation Theory in Transport and Diffusion Problems*. Pergamon Press, Oxford, UK, 1965.

EE Lewis and WF Jr. Miller. *Computational Methods of Neutron Transport*. American Nuclear Society, La Grange Park, IL, 1993.

RC Little, T Kawano, GD Hale, MT Pigni, Herman M, P Obložinský, ML Williams, ME Dunn, G Arbanas, D Wiarda, RD McKnight, JN McKamy, and JR Felty. Low-fidelity covariance project. *Nuclear Data Sheets*, 109(12):2828–2833, 2008. Special Issue on Workshop on Neutron Cross Section Covariances June 24-28, 2008, Port Jefferson, New York, USA.

RE MacFarlane. NJOY99.0 — code system for producing pointwise and multigroup neutron and photon cross sections from ENDF/B data. Technical Report PSR-480, LANL, Los Alamos, NM, 1994.

MS McKinley, B Beck, and DP McNabb. Nuclear and atomic database system (nads). `http://nuclear.llnl.gov/CNP/nads/NADSApplet.html`, July 2004.

V McLane and Members of the CSEWG. ENDF-201 4th edition supplement i: ENDF/B-VI summary documentation. Technical Report BNL-NCS-17641, BNL, Upton, NY, 1996.

EI Moses, T Diaz de la Rubia, E Storm, JF Latkowski, JC Farmer, RP Abbott, KJ Kramer, PF Peterson, HF Shaw, and RF Lehman II. A sustainable nuclear fuel cycle based on laser inertial fusion-fission energy (LIFE). *Fusion Science and Technology*, 56(2):547–565, August 2009. Proceedings of the Eighteenth Topical Meeting on the Technology of Fusion Energy (TOFE) (Part 2).

Atomic Nuclear and X-1-NAD EOS Data. New ace-formatted neutron and proton libraries based on ENDF/B-VII.0. Technical Report LA-UR-08-1999, LANL, Los Alamos, NM, May 2008.

Organization for Economic Co-Operation and Development — Nuclear Energy Agency: OECD/NEA. Joint evaluated fission and fusion file: JEFF-3.1.1. `http://www.nea.fr/dbforms/data/eva/evatapes/jeff_31/index-JEFF3.1.1.html`, January 2011.

HJ Park, HJ Shim, and CH Kim. Uncertainty propagation in monte carlo depletion analysis. *Nuclear Science and Engineering*, 167:196–208, January 2011. `http://epubs.ans.org/?a=11669`.

DB Pelowitz, JW Durkee, JS Elson, ML Fensin, JS Hendricks, MR James, RC Johns, GW McKinney, SG Mashnik, JM Verbeke, LS Waters, and TA Wilcox. Mcnpx 2.7.e extensions. Technical Report LA-UR-11-01502, LANL, Los Alamos, NM, March 2011.

DI Poston and Trellue. User's manual, version 2.0 for monteburns, version 1.0. Technical Report LA-UR-99-4999, LANL, Los Alamos, NM, September 1999.

JJ Powers, RP Abbott, M Fratoni, KJ Kramer, JF Latkowski, and JE Seifried. Neutronics design of a thorium-fueled fission blanket for LIFE (laser inertial fusion-based energy). San Diego, CA, June 2010. American Nuclear Society. Presented at the 2010 International Congress on Advances in Nuclear Power Plants (ICAPP) Embedded Topical Conference.

S Powers. Variance of a function of a random variable. Private Communications, August 2010.

BT Rearden and ML Williams. Eigenvalue contribution estimator for sensitivity calculations with TSUNAMI-3D. *Proceedings of 8th ICNCS*, May 2007. Presented at the 2007 8th International Conference on Nuclear Criticality Safety.

BT Rearden, CM Perfetti, ML Williams, and LM Petrie. SCALE sensitivity calculations using contribution theory. *Proceedings of SNA + MC2010*, October 2010. Presented at the 2010 Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2010.

BT Rearden, ML Williams, MA Jessee, DE Mueller, and DA Wiarda. Sensitivity and uncertainty analysis capabilities and data in SCALE. *Nuclear Technology*, 174(2):236–288, May 2011.

J Renze. Inner product. `http://mathworld.wolfram.com/InnerProduct.html`, August 2011.

State Atomic Energy Corporation — State Scientific Center of the Russion Federation — Institude of Physics and Power Engineering: Rosatom. Encyclopedia of neutron data rusfond (russian library files evaluated neutron data) complete package background selection of assessments (translated from russian). Technical Report 02.434.11.5001, Rosatom, Moscow, Russia, 2006.

State Atomic Energy Corporation — State Scientific Center of the Russion Federation — Institude of Physics and Power Engineering: Rosatom. Russian file of evaluated nuclear data RUSFOND-2010. `http://www.ippe.ru/podr/abbn/english/libr/intr-rosfond.php`, January 2011.

A Santamarina, D Bernard, P Blaise, M Coste, A Courcelle, TD Huynh, C Jouanne, P Leconte, O Litaize, S Mengelle, G Noguère, JM Ruggiéri, O Sérot, J Tommasi, C Vaglio, and JF Vidal. *The JEFF-3.1.1 Nuclear Data Library*. OECD/NEA, Paris, France, 2009.

JE Seifried, RP Abbott, M Fratoni, KJ Kramer, JF Latkowski, PF Peterson, JJ Powers, and JM Taylor. Explicit uncertainty analysis for tritium breeding in a laser inertial fusion engine. *ANS Transactions*, 102(1):250–251, June 2010. Presented at the 2010 American Nuclear Society (ANS) Annual Meeting.

JE Seifried, M Fratoni, KJ Kramer, JF Latkowski, PF Peterson, JJ Powers, and JM Taylor. Adjoint-based uncertainty analysis for essential reactions in a laser inertial fusion engine. *Fusion Science and Technology*, 60(2):692–697, August 2011. Proceedings of the Nineteenth Topical Meeting on the Technology of Fusion Energy (TOFE) (Part 2).

JE Seifried. Thermal modeling and feedback requirements for LIFE neutronic simulations. Technical Report LLNL-TR-416167, LLNL, August 2009. `http://dx.doi.org/10.2172/966912`.

DS Selengut. Variational analysis of a multi-dimensional system. Technical Report HW-59126, HNR, Hanford, WA, 1959.

K Shibata, T Kawano, T Nakagawa, O Iwamoto, J Katakura, T Fukahori, S Chiba, A Hasegawa, T Murata, H Matsunobu, T Ohsawa, Y Nakajima, T Yoshida, A Zukeran, M Kawai, M Baba, M Ishikawa, T Asami, T Watanabe, Y Watanabe, M Igashira, N Yamamuro, H Kitazawa, N Yamano, and H Takano. Japanese evaluated nuclear data library version 3 revision-3: JENDL-3.3. *Journal of Nuclear Science and Technology*, 39(11):1125–1136, 2002.

K Shibata, O Iwamoto, T Nakagawa, N Iwamoto, A Ichihara, S Kunieda, S Chiba, K Furutaka, N Otuka, T Ohsawa, H Matsunobu, A Zukeran, S Kamada, and J Katakura. JENDL-4.0: a new library for nuclear science and engineering. *Journal of Nuclear Science and Technology*, 48(1):1–30, 2011.

WM Jr. Stacey. *Variational Methods in Nuclear Reactor Physics*. Academic Press, New York, NY, 1974.

RJJ Stamm'ler and MJ Abbate. *Methods of Steady-State Reactor Physics in Nuclear Design*. Academic Press, London, UK, 1983.

TA Taiwo, G Palmiotti, G Aliberti, M Salvatores, and TK Kim. Uncertainty and target accuracy studies for the very high temperature reactor (VHTR) physics parameters. Technical Report ANL-GenIV-051, ANL, Argonne, IL, August 2005.

ORNL SCALE Team. Scale: A comprehensive modeling and simulation suite for nuclear safety analysis and design. Technical Report ORNL/TM-2005/39, ORNL, Oak Ridge, TN, June 2011. Available from Radiation Safety Information Computational Center at Oak Ridge National Laboratory as CCC-785.

H Trellue. `njoy-trellue.inp`. Private Communications, May 2010.

JL Vujic, E Greenspan, WE Kastenberg, Y Karni, and D Regev. Optimal neutron source & beam shaping assembly for boron neutron capture therapy. Technical Report UCBNE 4239, UCB, Berkeley, CA, April 2003.

JC Wagner, EL II Redmond, SP Palmtag, and JS Hendricks. MCNP: Multigroup/adjoint capabilities. Technical Report LA-12704 — UC-705, LANL, Los Alamos, NM, April 1994.

CR Weisbin, G de Saussure, RT Santoro, and D Gilai. Meeting cross-section requirements for nuclear-energy design. *Annals of Nuclear Energy*, 9(11-12):615–673, 1982. `http://dx.doi.org/10.1016/0306-4549(82)90006-8`.

EW Weisstein. Adjoint. `http://mathworld.wolfram.com/Adjoint.html`, August 2011.

EW Weisstein. Error propagation. http://mathworld.wolfram.com/ErrorPropagation.html, August 2011.

EW Weisstein. Solid of revolution. http://mathworld.wolfram.com/SolidofRevolution.html, August 2011.

EW Weisstein. Tree. http://mathworld.wolfram.com/Tree.html, August 2011.

Wikipedia. Taylor expansions for the moments of functions of random variables — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Taylor_expansions_for_the_moments_of_functions_of_random_variables&oldid=403047885, August 2011.

ML Williams. *Perturbation Theory for Nuclear Reactor Analysis*. CRC Press, Boca Raton, FL, 1982.

X-5 Monte Carlo Team. Mcnp – a general monte carlo n-particle transport code, version 5 — volume i: Overview and transport. Technical Report LA-UR-03-1987, LANL, Los Alamos, NM, October 2005.

A Yamamoto and N Sugimura. Improvement on multi-group scattering matrix in thermal energy range generated by NJOY. *Annals of Nuclear Energy*, 33:555–559, 2006. http://dx.doi.org/10.1016/j.anucene.2006.01.005.

G Zhigang, Y Hongwei, Z Youxiang, L Tingjin, Z Jingshang, L Ping, H Xiaolong, Z Zhixiang, and X Haihong. The updated version of the chinese evaluated nuclear data library (CENDL-3.1) and china nuclear data evaluation activities. *ND2007*, pages 753–757, 2007.

Appendix A

Source libraries for nuclear data covariances

Isotope	Library
^{1}H	Low Fidelity
^{2}H	Low Fidelity‡
^{3}H	Low Fidelity‡
^{3}He	Low Fidelity‡
^{4}He	Low Fidelity‡
^{6}Li	Low Fidelity
^{7}Li	ENDF/B-VII.0
^{7}Be	Low Fidelity
^{9}Be	Low Fidelity‡
^{10}B	Low Fidelity‡
^{11}B	Low Fidelity‡
^{nat}C	Low Fidelity‡
^{14}N	Low Fidelity‡
^{15}N	Low Fidelity‡
^{16}O	Low Fidelity‡
^{17}O	Low Fidelity
^{19}F	Low Fidelity‡
^{22}Na	Low Fidelity
^{23}Na	Low Fidelity
^{24}Mg	Low Fidelity
^{25}Mg	Low Fidelity
^{26}Mg	Low Fidelity
^{27}Al	Low Fidelity
^{28}Si	Low Fidelity
^{29}Si	Low Fidelity

continued on next page

continued from previous page

Isotope	Library
^{30}Si	Low Fidelity
^{31}P	Low Fidelity
^{32}S	Low Fidelity
^{33}S	Low Fidelity
^{34}S	Low Fidelity
^{36}S	Low Fidelity
^{35}Cl	Low Fidelity
^{37}Cl	Low Fidelity
^{36}Ar	Low Fidelity
^{38}Ar	Low Fidelity
^{40}Ar	Low Fidelity
^{39}K	Low Fidelity
^{40}K	Low Fidelity
^{41}K	Low Fidelity
^{40}Ca	Low Fidelity
^{42}Ca	Low Fidelity
^{43}Ca	Low Fidelity
^{44}Ca	Low Fidelity
^{46}Ca	Low Fidelity
^{48}Ca	Low Fidelity
^{45}Sc	Low Fidelity
^{46}Ti	Low Fidelity
^{47}Ti	Low Fidelity
^{48}Ti	Low Fidelity
^{49}Ti	Low Fidelity
^{50}Ti	Low Fidelity
^{nat}V	JENDL-3.3
^{51}V	Low Fidelity
^{50}Cr	Low Fidelity
^{52}Cr	Low Fidelity
^{53}Cr	Low Fidelity
^{54}Cr	Low Fidelity
^{55}Mn	Low Fidelity
^{54}Fe	Low Fidelity
^{56}Fe	Low Fidelity
^{57}Fe	Low Fidelity
^{58}Fe	Low Fidelity
^{58}Co	Low Fidelity
^{58m}Co	Low Fidelity
^{59}Co	Low Fidelity
^{58}Ni	Low Fidelity
^{59}Ni	Low Fidelity

continued on next page

continued from previous page

Isotope	Library
^{60}Ni	Low Fidelity
^{61}Ni	Low Fidelity
^{62}Ni	Low Fidelity
^{64}Ni	Low Fidelity
^{63}Cu	Low Fidelity
^{65}Cu	Low Fidelity
^{64}Zn	Low Fidelity
^{66}Zn	Low Fidelity
^{67}Zn	Low Fidelity
^{68}Zn	Low Fidelity
^{70}Zn	Low Fidelity
^{69}Ga	Low Fidelity
^{71}Ga	Low Fidelity
^{70}Ge	Low Fidelity
^{72}Ge	Low Fidelity
^{73}Ge	Low Fidelity
^{74}Ge	Low Fidelity
^{76}Ge	Low Fidelity
^{74}As	Low Fidelity
^{75}As	Low Fidelity
^{74}Se	Low Fidelity
^{76}Se	Low Fidelity
^{77}Se	Low Fidelity
^{78}Se	Low Fidelity
^{79}Se	Low Fidelity
^{80}Se	Low Fidelity
^{82}Se	Low Fidelity
^{79}Br	Low Fidelity
^{81}Br	Low Fidelity
^{78}Kr	Low Fidelity
^{80}Kr	Low Fidelity
^{82}Kr	Low Fidelity
^{83}Kr	Low Fidelity
^{84}Kr	Low Fidelity
^{85}Kr	Low Fidelity
^{86}Kr	Low Fidelity
^{85}Rb	Low Fidelity
^{86}Rb	Low Fidelity
^{87}Rb	Low Fidelity
^{84}Sr	Low Fidelity
^{86}Sr	Low Fidelity
^{87}Sr	Low Fidelity

continued on next page

continued from previous page

Isotope	Library
^{88}Sr	Low Fidelity
^{89}Sr	Low Fidelity
^{90}Sr	Low Fidelity
^{89}Y	Low Fidelity
^{90}Y	Low Fidelity
^{91}Y	Low Fidelity
^{90}Zr	Low Fidelity
^{91}Zr	Low Fidelity
^{92}Zr	Low Fidelity
^{93}Zr	Low Fidelity
^{94}Zr	Low Fidelity
^{95}Zr	Low Fidelity
^{96}Zr	Low Fidelity
^{93}Nb	Low Fidelity
^{94}Nb	Low Fidelity
^{95}Nb	Low Fidelity
^{92}Mo	Low Fidelity
^{94}Mo	Low Fidelity
^{95}Mo	Low Fidelity
^{96}Mo	Low Fidelity
^{97}Mo	Low Fidelity
^{98}Mo	Low Fidelity
^{99}Mo	Low Fidelity
^{100}Mo	Low Fidelity
^{99}Tc	Low Fidelity
^{96}Ru	Low Fidelity
^{98}Ru	Low Fidelity
^{99}Ru	Low Fidelity
^{100}Ru	Low Fidelity
^{101}Ru	Low Fidelity
^{102}Ru	Low Fidelity
^{103}Ru	Low Fidelity
^{104}Ru	Low Fidelity
^{105}Ru	Low Fidelity
^{106}Ru	Low Fidelity
^{103}Rh	Low Fidelity
^{105}Rh	Low Fidelity
^{102}Pd	Low Fidelity
^{104}Pd	Low Fidelity
^{105}Pd	Low Fidelity
^{106}Pd	Low Fidelity
^{107}Pd	Low Fidelity

continued on next page

continued from previous page

Isotope	Library
^{108}Pd	Low Fidelity
^{110}Pd	Low Fidelity
^{107}Ag	Low Fidelity
^{109}Ag	Low Fidelity
^{110m}Ag	Low Fidelity
^{111}Ag	Low Fidelity
^{106}Cd	Low Fidelity
^{108}Cd	Low Fidelity
^{110}Cd	Low Fidelity
^{111}Cd	Low Fidelity
^{112}Cd	Low Fidelity
^{113}Cd	Low Fidelity
^{114}Cd	Low Fidelity
^{115m}Cd	Low Fidelity
^{116}Cd	Low Fidelity
^{113}In	Low Fidelity
^{115}In	Low Fidelity
^{112}Sn	Low Fidelity
^{113}Sn	Low Fidelity
^{114}Sn	Low Fidelity
^{115}Sn	Low Fidelity
^{116}Sn	Low Fidelity
^{117}Sn	Low Fidelity
^{118}Sn	Low Fidelity
^{119}Sn	Low Fidelity
^{120}Sn	Low Fidelity
^{122}Sn	Low Fidelity
^{123}Sn	Low Fidelity
^{124}Sn	Low Fidelity
^{125}Sn	Low Fidelity
^{126}Sn	Low Fidelity
^{121}Sb	Low Fidelity
^{123}Sb	Low Fidelity
^{124}Sb	Low Fidelity
^{125}Sb	Low Fidelity
^{126}Sb	Low Fidelity
^{120}Te	Low Fidelity
^{122}Te	Low Fidelity
^{123}Te	Low Fidelity
^{124}Te	Low Fidelity
^{125}Te	Low Fidelity
^{126}Te	Low Fidelity

continued on next page

continued from previous page

Isotope	Library
^{127m}Te	Low Fidelity
^{128}Te	Low Fidelity
^{129m}Te	Low Fidelity
^{130}Te	Low Fidelity
^{132}Te	Low Fidelity
^{127}I	Low Fidelity
^{129}I	Low Fidelity
^{130}I	Low Fidelity
^{131}I	Low Fidelity
^{135}I	Low Fidelity
^{123}Xe	Low Fidelity
^{124}Xe	Low Fidelity
^{126}Xe	Low Fidelity
^{128}Xe	Low Fidelity
^{129}Xe	Low Fidelity
^{130}Xe	Low Fidelity
^{131}Xe	Low Fidelity
^{132}Xe	Low Fidelity
^{133}Xe	Low Fidelity
^{134}Xe	Low Fidelity
^{135}Xe	Low Fidelity
^{136}Xe	Low Fidelity
^{133}Cs	Low Fidelity
^{134}Cs	Low Fidelity
^{135}Cs	Low Fidelity
^{136}Cs	Low Fidelity
^{137}Cs	Low Fidelity
^{130}Ba	Low Fidelity
^{132}Ba	Low Fidelity
^{133}Ba	Low Fidelity
^{134}Ba	Low Fidelity
^{135}Ba	Low Fidelity
^{136}Ba	Low Fidelity
^{137}Ba	Low Fidelity
^{138}Ba	Low Fidelity
^{140}Ba	Low Fidelity
^{138}La	Low Fidelity
^{139}La	Low Fidelity
^{140}La	Low Fidelity
^{136}Ce	Low Fidelity
^{138}Ce	Low Fidelity
^{139}Ce	Low Fidelity

continued on next page

continued from previous page

Isotope	Library
^{140}Ce	Low Fidelity
^{141}Ce	Low Fidelity
^{142}Ce	Low Fidelity
^{143}Ce	Low Fidelity
^{144}Ce	Low Fidelity
^{141}Pr	Low Fidelity
^{142}Pr	Low Fidelity
^{143}Pr	Low Fidelity
^{142}Nd	Low Fidelity
^{143}Nd	Low Fidelity
^{144}Nd	Low Fidelity
^{145}Nd	Low Fidelity
^{146}Nd	Low Fidelity
^{147}Nd	Low Fidelity
^{148}Nd	Low Fidelity
^{150}Nd	Low Fidelity
^{147}Pm	Low Fidelity
^{148}Pm	Low Fidelity
^{148m}Pm	Low Fidelity
^{149}Pm	Low Fidelity
^{151}Pm	Low Fidelity
^{144}Sm	Low Fidelity
^{147}Sm	Low Fidelity
^{148}Sm	Low Fidelity
^{149}Sm	Low Fidelity
^{150}Sm	Low Fidelity
^{151}Sm	Low Fidelity
^{152}Sm	Low Fidelity
^{153}Sm	Low Fidelity
^{154}Sm	Low Fidelity
^{151}Eu	Low Fidelity
^{152}Eu	Low Fidelity
^{153}Eu	Low Fidelity
^{154}Eu	Low Fidelity
^{155}Eu	Low Fidelity
^{156}Eu	Low Fidelity
^{157}Eu	Low Fidelity
^{152}Gd	Low Fidelity
^{153}Gd	Low Fidelity
^{154}Gd	Low Fidelity
^{155}Gd	Low Fidelity
^{156}Gd	Low Fidelity

continued on next page

continued from previous page

Isotope	Library
^{157}Gd	Low Fidelity
^{158}Gd	Low Fidelity
^{160}Gd	Low Fidelity
^{159}Tb	Low Fidelity
^{160}Tb	Low Fidelity
^{156}Dy	Low Fidelity
^{158}Dy	Low Fidelity
^{160}Dy	Low Fidelity
^{161}Dy	Low Fidelity
^{162}Dy	Low Fidelity
^{163}Dy	Low Fidelity
^{164}Dy	Low Fidelity
^{165}Ho	Low Fidelity
^{166m}Ho	Low Fidelity
^{162}Er	Low Fidelity
^{164}Er	Low Fidelity
^{166}Er	Low Fidelity
^{167}Er	Low Fidelity
^{168}Er	Low Fidelity
^{170}Er	Low Fidelity
^{175}Lu	Low Fidelity
^{176}Lu	Low Fidelity
^{174}Hf	Low Fidelity
^{176}Hf	Low Fidelity
^{177}Hf	Low Fidelity
^{178}Hf	Low Fidelity
^{179}Hf	Low Fidelity
^{180}Hf	Low Fidelity
^{181}Ta	Low Fidelity
^{182}Ta	Low Fidelity
^{182}W	Low Fidelity
^{183}W	Low Fidelity
^{184}W	Low Fidelity
^{186}W	Low Fidelity
^{185}Re	Low Fidelity
^{187}Re	Low Fidelity
^{191}Ir	Low Fidelity
^{193}Ir	Low Fidelity
^{197}Au	Low Fidelity
^{196}Hg	Low Fidelity
^{198}Hg	Low Fidelity
^{199}Hg	Low Fidelity

continued on next page

continued from previous page

Isotope	Library
^{200}Hg	Low Fidelity
^{201}Hg	Low Fidelity
^{202}Hg	Low Fidelity
^{204}Hg	Low Fidelity
^{204}Pb	Low Fidelity
^{206}Pb	Low Fidelity
^{207}Pb	Low Fidelity
^{208}Pb	Low Fidelity
^{209}Bi	Low Fidelity
^{225}Ac	Low Fidelity‡
^{226}Ac	Low Fidelity‡
^{227}Ac	Low Fidelity‡
^{227}Th	Low Fidelity‡
^{228}Th	Low Fidelity‡
^{229}Th	Low Fidelity‡
^{230}Th	Low Fidelity‡
^{232}Th	JENDL-4.0
^{233}Th	Low Fidelity‡
^{234}Th	Low Fidelity‡
^{231}Pa	Low Fidelity‡
^{232}Pa	Low Fidelity‡
^{233}Pa	Low Fidelity‡
^{232}U	JENDL-4.0
^{233}U	JENDL-4.0
^{234}U	Low Fidelity
^{235}U	JENDL-4.0
^{236}U	Low Fidelity
^{237}U	Low Fidelity‡
^{238}U	JENDL-4.0
^{239}U	Low Fidelity‡
^{240}U	Low Fidelity‡
^{241}U	Low Fidelity‡
^{235}Np	Low Fidelity‡
^{236}Np	Low Fidelity‡
^{237}Np	Low Fidelity
^{238}Np	Low Fidelity‡
^{239}Np	Low Fidelity‡
^{236}Pu	Low Fidelity‡
^{237}Pu	Low Fidelity‡
^{238}Pu	Low Fidelity
^{239}Pu	JENDL-4.0
^{240}Pu	Low Fidelity

continued on next page

continued from previous page	
Isotope	Library
^{241}Pu	Low Fidelity
^{242}Pu	Low Fidelity
^{243}Pu	Low Fidelity‡
^{244}Pu	Low Fidelity‡
^{246}Pu	Low Fidelity‡
^{241}Am	Low Fidelity
^{242}Am	Low Fidelity‡
^{242m}Am	Low Fidelity
^{243}Am	Low Fidelity
^{244}Am	Low Fidelity‡
^{244m}Am	Low Fidelity‡
^{241}Cm	Low Fidelity‡
^{242}Cm	Low Fidelity
^{243}Cm	Low Fidelity
^{244}Cm	Low Fidelity
^{245}Cm	Low Fidelity
^{246}Cm	Low Fidelity‡
^{247}Cm	Low Fidelity‡
^{248}Cm	Low Fidelity‡
^{249}Cm	Low Fidelity‡
^{250}Cm	Low Fidelity‡
^{249}Bk	Low Fidelity‡
^{250}Bk	Low Fidelity‡
^{249}Cf	Low Fidelity‡
^{250}Cf	Low Fidelity‡
^{251}Cf	Low Fidelity‡
^{252}Cf	Low Fidelity‡
^{253}Cf	Low Fidelity‡
^{254}Cf	Low Fidelity‡
^{253}Es	Low Fidelity‡
^{254}Es	Low Fidelity‡
^{255}Es	Low Fidelity‡
^{255}Fm	Low Fidelity‡

Table A.1: Source libraries for nuclear data covariances

‡ denotes that a library was processed with the `NJOY99` `ERRORR` module with `ENDF/B-VII.0` nominal cross-sections.