LA-UR-17-31022
Approved for public release; distribution is unlimited.

Title: Sensitivity Analysis of Cf-252 (sf) Neutron and Gamma Observables in CGMF

Author(s):
Carter, Austin Lewis
Talou, Patrick
Stetcu, Ionel
Kiedrowski, Brian Christopher
Jaffke, Patrick John

Intended for: Report

Issued: 2017-12-06

Sensitivity Analysis of Cf-252(sf) Neutron and Gamma Observables in CGMF

Austin Carter[a,b], Patrick Talou [b], Ionel Stetcu [b], Patrick Jaffke [b], Brian Kiedrowski[a]

[a]*University of Michigan, Nuclear Engineering and Radiological Sciences, 2355 Bonisteel Blvd., 1906 Cooley Bldg., Ann Arbor, Michigan 48109*

[b]*Los Alamos National Laboratory, T-2 Nuclear Theory Group, Los Alamos, New Mexico 87545*

Abstract - *CGMF is a Monte Carlo code that simulates the decay of primary fission fragments by emission of neutrons and gamma rays, according to the Hauser-Feshbach equations. As the CGMF code was recently integrated into the MCNP6.2 transport code, great emphasis has been placed on providing optimal parameters to CGMF such that many different observables are accurately represented. Of these observables, the prompt neutron spectrum, prompt neutron multiplicity, prompt gamma spectrum, and prompt gamma multiplicity are crucial for accurate transport simulations of criticality and nonproliferation applications. This contribution to the ongoing efforts to improve CGMF presents a study of the sensitivity of various neutron and gamma observables to several input parameters for Californium-252 spontaneous fission. Among the most influential parameters are those that affect the input yield distributions in fragment mass and total kinetic energy (TKE). A new scheme for representing Y(A,TKE) was implemented in CGMF using three fission modes, S1, S2 and SL. The sensitivity profiles were calculated for 17 total parameters, which show that the neutron multiplicity distribution is strongly affected by the TKE distribution of the fragments. The total excitation energy (TXE) of the fragments is shared according to a parameter RT, which is defined as the ratio of the light to heavy initial temperatures. The sensitivity profile of the neutron multiplicity shows a second order effect of RT on the mean neutron multiplicity. A final sensitivity profile was produced for the parameter alpha, which affects the spin of the fragments. Higher values of alpha lead to higher fragment spins, which inhibit the emission of neutrons. Understanding the sensitivity of the prompt neutron and gamma observables to the many CGMF input parameters provides a platform for the optimization of these parameters.*

1. INTRODUCTION

Knowledge of correlations between fission observables is essential for high-fidelity simulations of the complex fission process where such correlations play an important role. These simulations are crucial for many applications including criticality safety, nuclear safeguards and non-proliferation, nuclear energy, nuclear threat reduction and response, radiation detection and measurement, radiation health protection, stockpile stewardship, etc. Hence, there has been a growing interest in fission science in recent years, demonstrated by increasing research in experimental fission facilities and detectors, and robust computer simulation tools to test theoretical fission models.

The Cascading Gamma ray Multiplicity with Fission (CGMF) code, developed at Los Alamos National Laboratory, is a state-of-the-art nuclear fission code that uses the Hauser-Feshbach statistical theory of nuclear reactions to simulate fission events. CGMF has recently been integrated into the MCNP6.2 transport code, providing correlations that were previously neglected in observables. In previous versions of MCNP, fission events were treated with ENDF/B nuclear data tables, with each distribution sampled independently. Thus, reaction channels and outgoing products were uncorrelated, seeing that multi-dimensional distributions are ignored.

1.A. CGMF Background Theory

CGMF provides a much more accurate treatment of fission events, however the code requires a significant level of optimization to reproduce experimental data. To initiate a fission event, CGMF samples fission fragment mass (A), charge (Z), Total Kinetic Energy (TKE), spin (J), and parity (π) from experimental data or systematics, depending on the variable. This paper demonstrates that responses of particular interest, those related to neutron and γ ray multiplicities and spectra, are highly sensitive to the yield distribution of mass and TKE, Y(A,TKE) (Sec. 5). Also shown are the effects of the parameter α, which influences the competition between emission of γ rays and neutrons through altering the spin distribution (Sec. 7). The total excitation energy of the fragments (TXE) is found with the Q-value of the reaction, the incident particle energy, the neutron binding energy, and the total kinetic energy (Sec. 6). The parameter R_T defines the sharing of the total excitation energy between fragments and has significant impacts on the neutron multiplicity and spectrum. After the initial fragment properties are sampled, CGMF performs a Monte Carlo simulation of the de-excitation of the fission fragments. Using the excitation energy of the fragments, the probabilities of neutron and γ ray emission are calculated. A specific decay path is chosen and the de-excitation proceeds in this manner until the fragments reach a ground or metastable state.

2. CGMF Output and Convergence

Upon completion of each fission event, CGMF records data for the event in a "histories-vectors.CGMF" file. An example of this file is shown in App. A. The first line of data for each event contains the light fragment mass, charge, excitation energy, spin, parity, kinetic energy, neutron multiplicity, γ ray multiplicity, and internal conversion multiplicity. The second line contains the fragment momentum vector, in center of mass frame and then lab frame. If the neutron multiplicity is non-zero, the third line contains each neutron's momentum vector and energy in center of mass frame and then lab frame. If the γ ray multiplicity is non-zero, the fourth line contains each γ ray momentum vector and energy likewise in center of mass frame and then lab frame. The same data is then shown for the heavy fragment, immediately after the corresponding light fragment.

Part of this work, in analyzing the sensitivities of specific responses to input parameters, was to produce an efficient method to compile the CGMF output data into a concise summary file. In the current configuration of the summary file, the data from the "histories-vectors.CGMF" is summarized with 20 observables. Those observables are presented in Sec. 5, and an example summary file is contained in App. B.

Another significant analysis of CGMF computation was finding a minimum number of fission events, for which a set of responses converge within a specified tolerance level. The Cf-252(sf) observables used in this analysis are the mean neutron multiplicity ($\bar{\nu}$), the mean γ ray multiplicity ($\overline{N}_{\gamma}$), each of their second and third factorial moments, as well as mean neutron and γ ray energies. Tolerances for each were selected according to their respective uncertainties according to Ref. 1. Figure 1 shows that after 50,000 events, all responses fall within the tolerances. Thus in future calculations, at least 50,000 events were simulated.

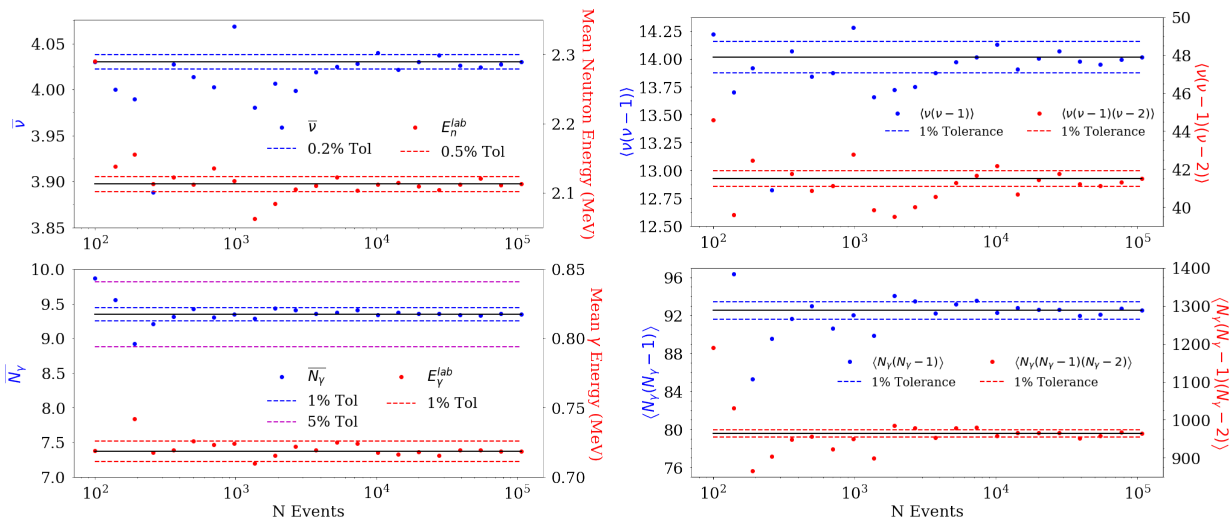

Fig. 1. Convergence criteria for eight CGMF responses

3. Cf-252(sf) Average Neutron Multiplicity ($\bar{\nu}$)

Reproducing the average neutron multiplicity is vitally important for criticality safety applications. As a first pass to doing sensitivity calculations with CGMF, an analysis was performed to analyze the effects of the average TKE ($\overline{TKE}$) on $\bar{\nu}$ for Cf-252(sf). This reaction was used because there exists a significant amount of good experimental data, and notably $\bar{\nu}$ has a very small evaluated uncertainty (0.13%). Figure 2 shows the current version of CGMF strongly over predicts $\bar{\nu}$, and there is a linear correlation between $\bar{\nu}$ and $\overline{TKE}$. To achieve $\bar{\nu} = 3.756$ (Ref. 1), $\overline{TKE} = 186.3\ MeV$, which is inside the evaluated uncertainty (Ref. 2). This analysis shows the strong effects the value of $\overline{TKE}$ has on $\bar{\nu}$.

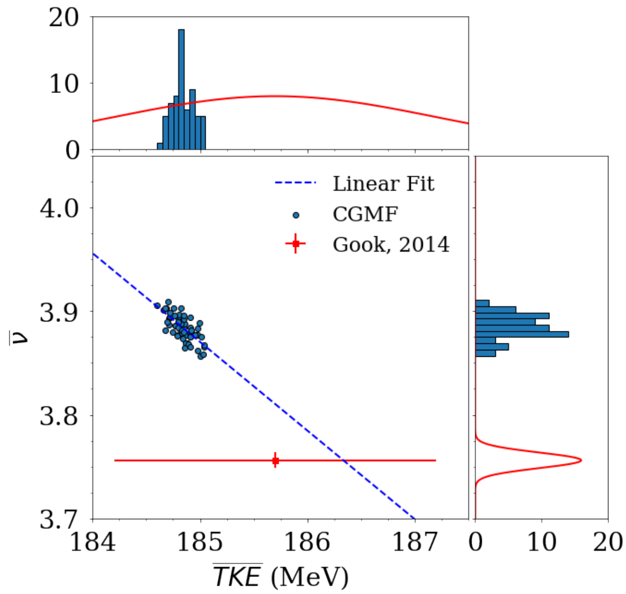

Fig. 2. Sensitivity of $\bar{\nu}$ on $\overline{TKE}$ for Cf-252(sf)

4. Brosa Modes Model for Y(A,TKE)

Previous versions of CGMF have relied on large, tabulated data files for the fission fragment yields in mass and TKE. Brosa et al. (Ref. 3) propose that Y(A) and Y(TKE) can be decomposed into modes or "fission channels, thereby creating a systematic way to sample the fragment mass and TKE. Applying this systematic method would allow for compound nuclei with unknown yield distributions to be sampled and the fragments to be propagated through CGMF. Thus CGMF can transition to become a predictive fission event generator for compound nuclei with no available fission data.

In the Brosa Modes model, the Y(A) distribution is decomposed into a sum of an arbitrary number of modes, where the mass yield of each mode is

$$Y_m(A) = \frac{w_m}{\sqrt{8\pi\sigma_{A,m}^2}}\left(exp\left(-\frac{A-\bar{A}_h^m}{2\sigma_{A,m}^2}\right) + exp\left(-\frac{A-A_{cn}+\bar{A}_h^m}{2\sigma_{A,m}^2}\right)\right). \qquad (1)$$

In Eq. 1, w_m is the weight of the mode, $\bar{A}_h^m$ is the mean heavy fragment mass, and $\sigma_{A,m}^2$ is the variance of the heavy side of the distribution. As the pre-neutron emission Y(A) distribution is perfectly symmetric around half the mass of the compound nucleus, Y(A) can be treated as a sum of two Gaussian distributions centered on $\bar{A}_h^m$ and $A_{cn} - \bar{A}_h^m$. The fragment mass distribution with component modes S1, S2, and SL is shown in Fig. 3. The S1 and S2 are standard asymmetric modes, and the SL mode is the Super Long symmetric mode. It can be seen using that the input parameters from Ref. 3 leads to significant deviations in the peaks and wings of the distribution. This effect is mitigated with a $\frac{\chi^2}{dof}$ optimization in section 4.A.

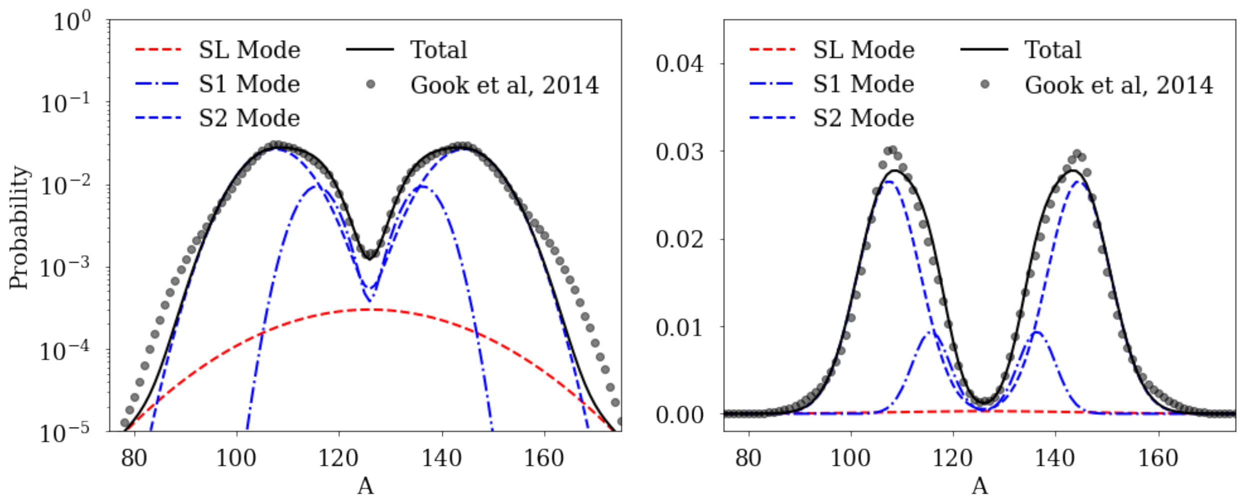

Fig. 3. Y(A) distribution for Cf-252(sf) decomposed into S1, S2 and SL modes

In order to produce the two dimensional Y(A,TKE), a distribution of TKE given a fragment mass was produced by Brosa et al (Ref. 3). The Y(TKE|A) distribution is not fit with Gaussian shapes, as the function is significantly skewed from a Gaussian at the tails. Instead, the TKE is a function of the distance between the centroids of the nascent fragments, D, according to

$$TKE = \frac{Z_l Z_h e^2}{D} \ , \tag{2}$$

where Z_l and Z_h are the charge numbers of the fragments. Using a shifted distance between centroids, $T = D - d_{min}$, and the uniform-charge-density relation, $Z = \left(\frac{Z_{cn}}{A_{cn}}\right) A$,

$$T_m(A) = \frac{\left(\frac{Z_{cn}}{A_{cn}}\right)^2 (A_{cn} - A) A e^2}{TKE} - d_{min}^m \ , \tag{3}$$

and the modal TKE|A distribution is

$$Y_m(TKE|A) = \left(\frac{200}{TKE}\right)^2 exp\left(2\frac{d_{max}^m - d_{min}^m}{d_{dec}^m} - \frac{T_m(A)}{d_{dec}^m} - \frac{(d_{max}^m - d_{min}^m)^2}{T_m(A)\, d_{dec}^m}\right) . \tag{4}$$

In this distribution, d_{max}^m is the half-length between fragment centroids with the most favorable potential energy to cause scission, d_{min}^m is the half-length below which scission will not occur, d_{dec}^m is the length constant describing the exponential decay of scission probability, and m denotes the mode. Then the yield in mass and TKE is

$$Y(A, TKE) = \sum_m Y_m(TKE|A)\, Y_m(A) \ . \tag{5}$$

With six parameters for each mode, w_m, $\bar{A}_h^m$, $\sigma_{A,m}$, d_{min}^m, d_{max}^m, d_{dec}^m, and three modes for Cf-252(sf), there are 18 parameters necessary to create Y(A,TKE). As the SL mode is symmetric about the half mass of the compound nucleus, $\bar{A}_h^{SL}$ remains constant at 126 and the number of adjustable parameters is reduced to 17. For the weights, the sum of the three weights is unity, however there are three possible combinations of adjusting one weight and compensating with another. Thus this constraint does not eliminate a degree of freedom.

4.A. Optimization of Y(A,TKE) Parameters to Fit Data

A sequential optimization scheme was implemented to better fit the Y(A,TKE) Brosa Modes model to experimental yield data (Ref. 1). The optimization tool used was scipy.optimize.minimize (Ref. 4) to minimize the $\frac{\chi^2}{dof}$ for the mass distribution, the TKE distribution, and Y(A,TKE). As this is a hyper-dimensional optimization problem, being fit to a very large data set, the optimization was broken into components to reduce computation time and to optimize parameters separately. The steps of the sequential optimization are shown in Table 1. It is important to note that in the first step, where all parameters are being optimized, a predefined number of iterations was selected, which severely constrained the optimization, and only slightly modified the most sensitive parameters. Thus, a minimal $\frac{\chi^2}{dof}$ was not found for this step.

TABLE 1

Y(A,TKE) Brosa Modes Sequential Optimization Scheme

Optimization Step	Updated Parameters	# Iterations
1. Minimize $\frac{\chi^2}{dof}$ for Y(A,TKE)	w_m, $\bar{A}_h^m$, $\sigma_{A,m}$, d_{min}^m, d_{max}^m, d_{dec}^m For all 3 modes	30
2. Minimize $\frac{\chi^2}{dof}$ for Y(A) using S2 mode	w_{S2}, $\bar{A}_h^{S2}$, $\sigma_{A,S2}$	3
3. Minimize $\frac{\chi^2}{dof}$ for Y(A) using S1 mode	w_{S1}, $\bar{A}_h^{S1}$, $\sigma_{A,S1}$	4
4. Minimize $\frac{\chi^2}{dof}$ for Y(A) using SL mode	w_{SL}, $\bar{A}_h^{SL}$, $\sigma_{A,SL}$	6
5. Minimize $\frac{\chi^2}{dof}$ for Y(TKE) using S2 mode	w_{S2}, d_{min}^{S2}, d_{max}^{S2}, d_{dec}^{S2}	13
6. Minimize $\frac{\chi^2}{dof}$ for Y(TKE) using S1 mode	w_{S1}, d_{min}^{S1}, d_{max}^{S1}, d_{dec}^{S1}	9
7. Minimize $\frac{\chi^2}{dof}$ for Y(TKE) using SL mode	w_{SL}, d_{min}^{SL}, d_{max}^{SL}, d_{dec}^{SL}	13
8. Minimize $\frac{\chi^2}{dof}$ for Y(A,TKE)	w_{S2}, w_{S1}, w_{SL}	14

The optimized values for the parameters are shown in Table 2. A comparison between the experimental data, Brosa Modes fit and the residuals for Y(A,TKE) is shown in Fig. 4. Finally, the fit for Y(A) and Y(TKE) is shown in Fig. 5. Figure 5 demonstrates the considerable improvements with the optimized parameters.

TABLE 2

Brosa Modes Optimized Parameters

Mode	w_m	$\bar{A}_h^m$	$\sigma_{A,m}$	d_{min}^m	d_{max}^m	d_{dec}^m
S2	0.813	144.5	6.11	14.5	18.2	0.446
S1	0.173	136.3	3.70	10.9	17.3	0.135
SL	0.014	126.0	18.2	16.7	20.4	0.166

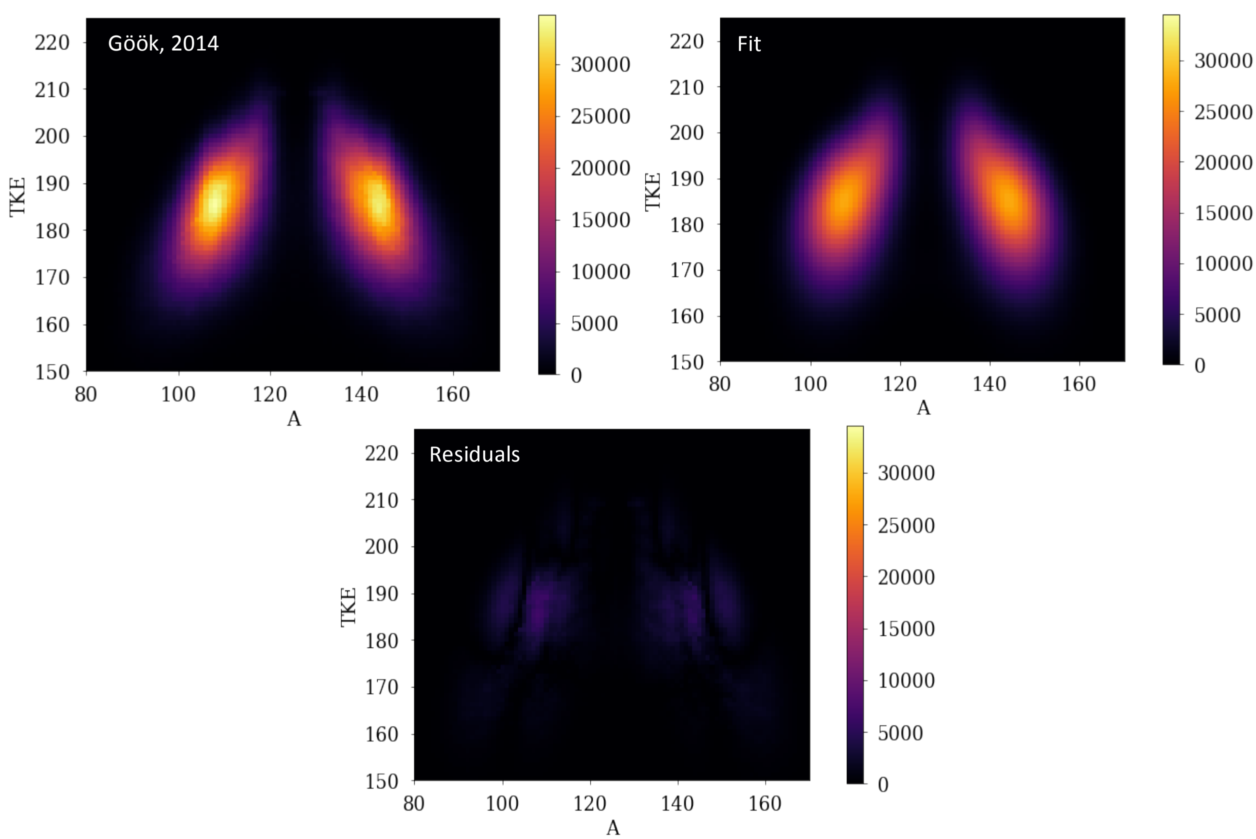

Fig. 4. Comparison of Y(A,TKE) for Brosa Modes Optimized Parameter Fit

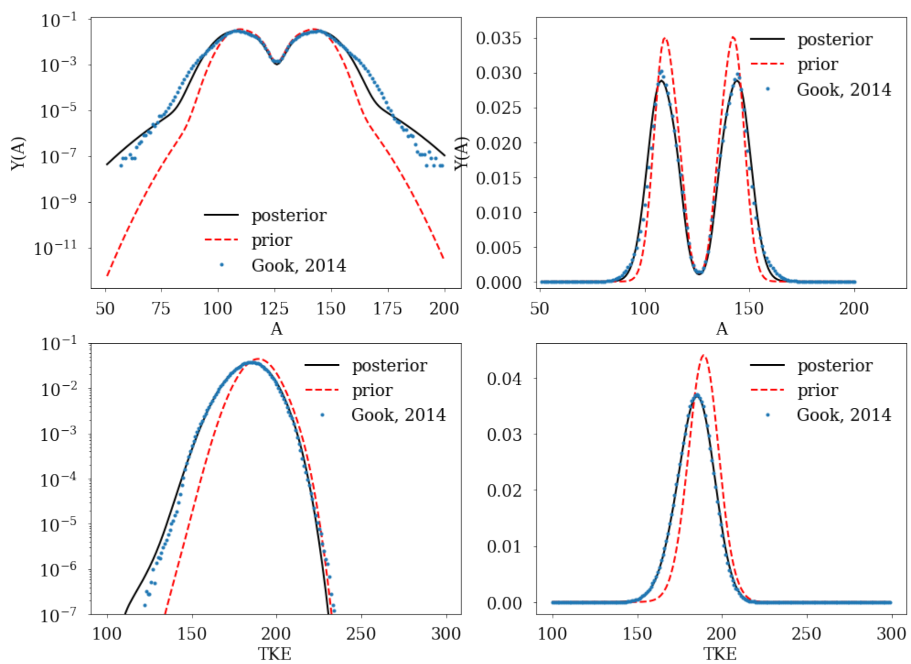

Fig. 5. Y(A) and Y(TKE) comparison of Brosa Modes fit to Göök et al. data

5. Sensitivity Analysis of CGMF Responses to Brosa Mode Parameters

A new C++ class for the sampling of fragment mass and kinetic energy was created for the Brosa Mode model, and called "BrosaYields". The important methods of this class are the "readfile" class, which reads a file from the data directory (specified when running executable with –y flag), and stores the parameter values. An example of an acceptable input file for the "BrosaYields" class is found in App. C. The new class has a method to sample a mode based on the weights of the modes. The "sampleA" method samples the heavy fragment mass according to

$$A = \bar{A}_h^m + \sigma_{A,m}\sqrt{-2\log(\xi_1)}\cos(2\pi\xi_2) \ , \tag{6}$$

where the mode is passed as an argument and ξ_1 and ξ_2 are uniform pseudo-random numbers from 0 to 1. The "sampleTKEA" method accepts a mode and a value for fragment mass and samples TKE based on the TKE probabilities for that mass, which is generated in the constructor. The "sampleZA" method uses Wahl systematics to sample the fragment charge from its mass (Ref. 5). Appendix D contain the source code for the "BrosaYields" class.

A sensitivity analysis was performed on 20 CGMF responses with respect to the 17 adjustable parameters ($\bar{A}^{SL}$ remains 126.0 as this mode is symmetric). The analysis is based on obtaining the relative sensitivity coefficients for all responses and all parameters of the Brosa Modes model. The sensitivity coefficient describes the relative change in response R to the change in parameter x, and is defined by

$$S_{R,x} = \frac{x}{R}\frac{\partial R}{\partial x} \ . \tag{7}$$

Fig. 6 shows the intensity of the sensitivity coefficients. The 20 responses are the average neutron and γ ray energies in lab and center of mass frame and the standard deviation, the average multiplicities of γ rays and neutrons and their 2nd and 3rd factorial moments, the average spin, average TKE, and average TXE, each with one standard deviation. The sensitivity coefficients for each response were obtained with a brute force method. A linearly spaced array of one parameter was provided to CGMF, while all other parameters were held at their nominal, optimized values. Each array of linearly spaced parameters contains the nominal value of the parameter as well as 10 values that extend to approximately $\pm$30% of the nominal value. There are several cases where the limits of the parameters were intuitively constrained, such as ensuring $d_{max}^{m} > d_{min}^{m}$, $w_m > 0$, and $\bar{A}_h^m$ for all modes. The responses were then gathered and a linear approximation was made for the partial derivative in Eq. 7. This analysis shows that the most influential parameters in the Brosa Modes model are d_{max}^{S2}, $\bar{A}_h^{S2}$, d_{max}^{S1}, and w_{S2} as these columns have the strongest sensitivity coefficients.

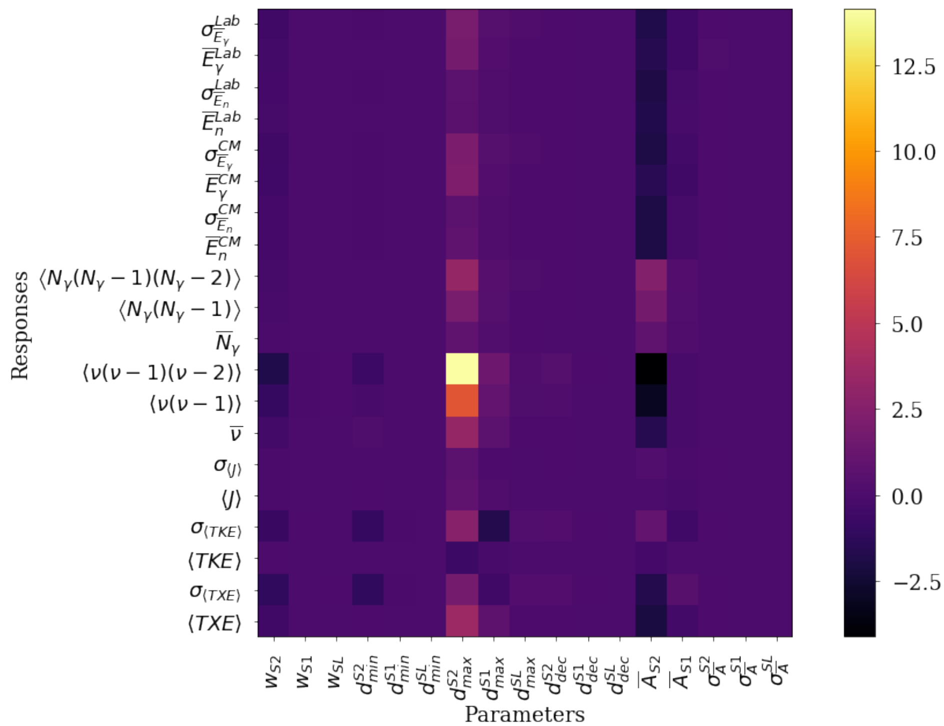

Fig. 6. Sensitivity Coefficients of Brosa Modes Model

5.A. Adding Constraints to Brosa Modes Parameters Using Sensitivity Coefficients

Seeing that d_{max}^{S2}, $\bar{A}_h^{S2}$, d_{max}^{S1}, and w_{S2} are the most sensitive parameters, constraints can be placed on these parameters such that key observables are constrained. The analysis in Sec. 3 demonstrates the strong effects of the TKE of the fragments on $\bar{\nu}$, thus constraints can be placed on d_{max}^{S2}, $\bar{A}_h^{S2}$, d_{max}^{S1}, and w_{S2} to restrict $\bar{\nu}$ to 3.76 ± 0.15 and $\langle TKE \rangle$ to $185.7 \pm 1.5\ MeV$ for Cf-252(sf) (Ref. 1). Figure 7 shows the constraints that are applicable to the parameters, which are necessary for further use of the Brosa Modes model in analyzing the spontaneous fission of Cf-252, and optimization of the parameters.

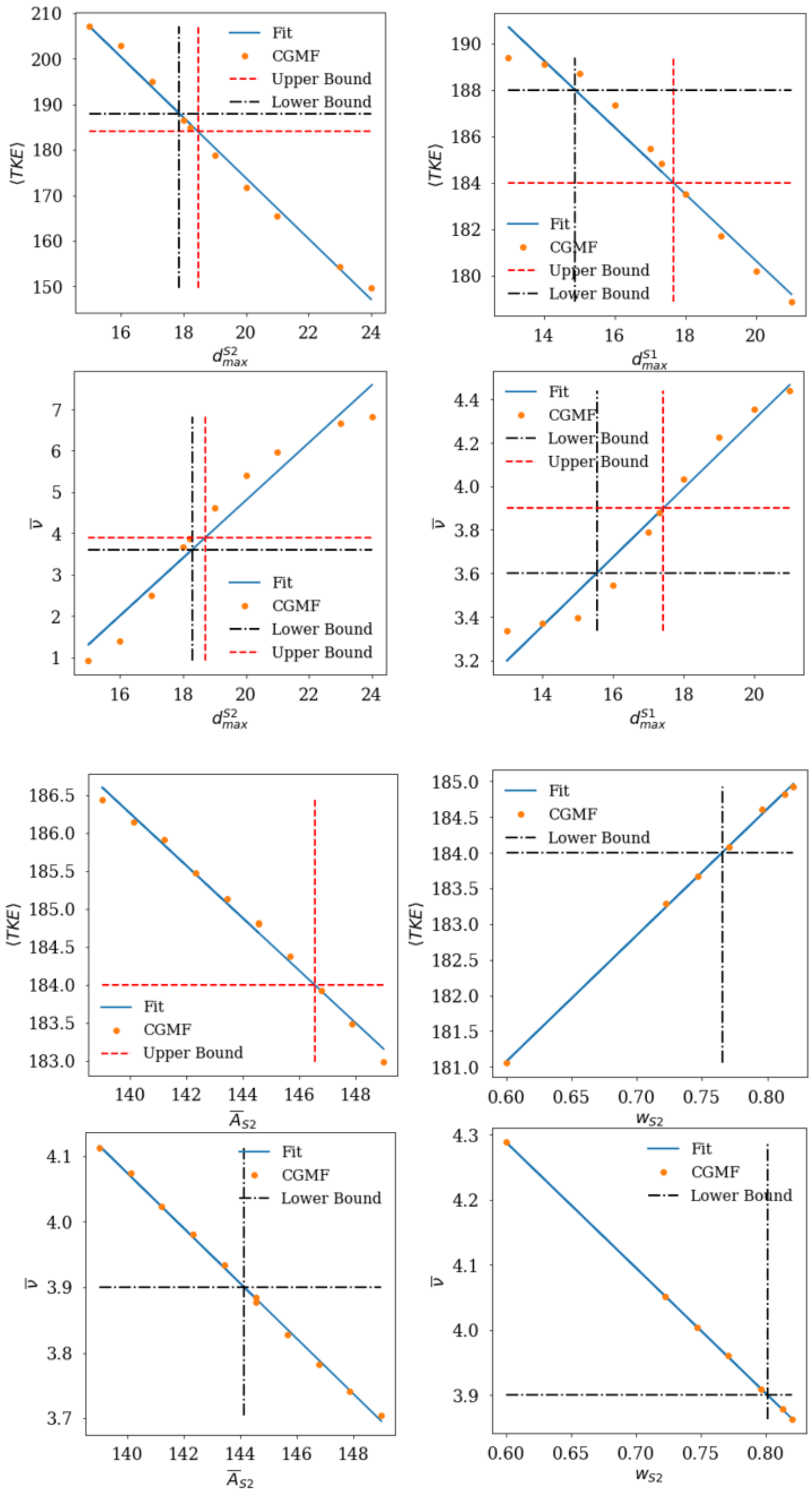

Fig. 7. Constraints on d_{max}^{S2}, $\bar{A}_h^{S2}$, d_{max}^{S1}, and w_{S2} applied to $\bar{\nu}$ and $\langle TKE \rangle$

6. Sensitivity Analysis of R_T

The parameter R_T accounts for the uneven split of excitation energy between the nascent fission fragments. As this sharing is not a directly measureable quantity, a theoretical ratio of initial fragment temperatures is built into CGMF, where

$$R_T = \frac{T_0^l}{T_0^h} \ . \tag{8}$$

After sampling Y(A,Z,TKE), the TXE of the fragments is

$$TXE = Q_f + E_n^{inc} + B_n(A_c, Z_c) - TKE \ , \tag{9}$$

where Q_f is the Q-value of the reaction,

$$Q_f = M_n(A_l, Z_l) + M_n(A_h, Z_h) - M_n(A_c,\ Z_c) \ , \tag{10}$$

M_n refers to the nuclear mass, E_n^{inc} is the kinetic energy of the incident neutron, and $B_n(A_c, Z_c)$ is the neutron binding energy of the compound nucleus. The TXE is then dissipated through emission of neutron and γ rays, until a ground or metastable state is reached.

An array of R_T values was used as CGMF input to analyze the sensitivity of several responses to various R_T. Figure 8 shows the average neutron and gamma multiplicity as a function of fragment mass for various R_T. The shape of $\bar{\nu}(A)$ changes dramatically with R_T. Low R_T corresponds to very little excitation energy in the light fragment, thus very few neutrons are emitted by the light fragment. There are no significant effects on $\overline{N_\gamma}(A)$ due to R_T.

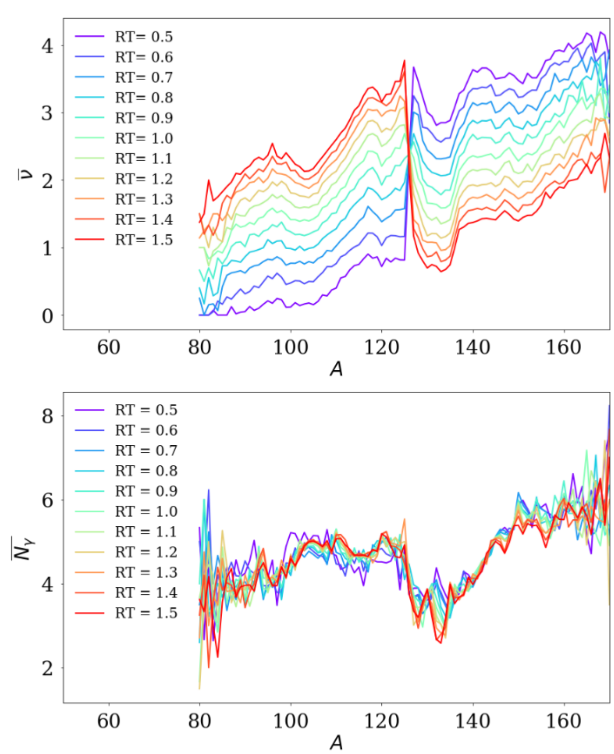

Fig. 8. Neutron and Gamma multiplicities as functions of fragment mass with various R_T

Further study of the neutron multiplicity shows that $\bar{\nu}$ has a nonlinear correlation to R_T. Figure 9 shows this 2nd order correlation, where the fit is characterized by

$$\bar{\nu}(R_T) = -0.505R_T^2 + 1.034R_T + 3.527 \ . \tag{11}$$

From Eq. 11, to achieve the experimental neutron multiplicity for Cf-252(sf) $\bar{\nu} = 3.76$, then $R_T = 3.86$, which would drive a very large majority of the excitation energy to the light fragment, and produce an unrealistic excitation energy sharing. This analysis was performed with experimental Y(A,Z,TKE) from Brosa et al. (Ref. 3). To accurately reproduce $\bar{\nu}$, the average TKE of the fragments must be optimized before constraining R_T (Sec. 3).

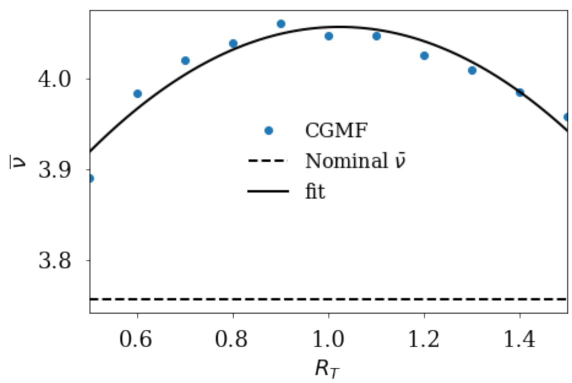

Fig. 9. Sensitivity of $\bar{\nu}$ to R_T

7. Sensitivity Analysis of α

The Hauser-Feshbach equations require knowledge of the initial fragment spin to perform a de-excitation cascade. The fragment spin distribution is controlled by the parameter α, where the J^{π} initial distribution is

$$\rho(J,\pi) = \frac{1}{2}(2J+1)\, exp\left(-\frac{J(J+1)\hbar^2}{2\alpha T I_0(A,Z)}\right) . \tag{12}$$

Similar analyses were performed with α, as those with R_T. Figure 10 shows the average neutron and gamma multiplicity as a function of fragment mass for various α. A higher value of alpha leads to high fragment spins, which inhibits neutron emission. Thus the $\bar{\nu}(A)$ decreases with increasing α, and $\bar{N}_{\gamma}(A)$ increases dramatically.

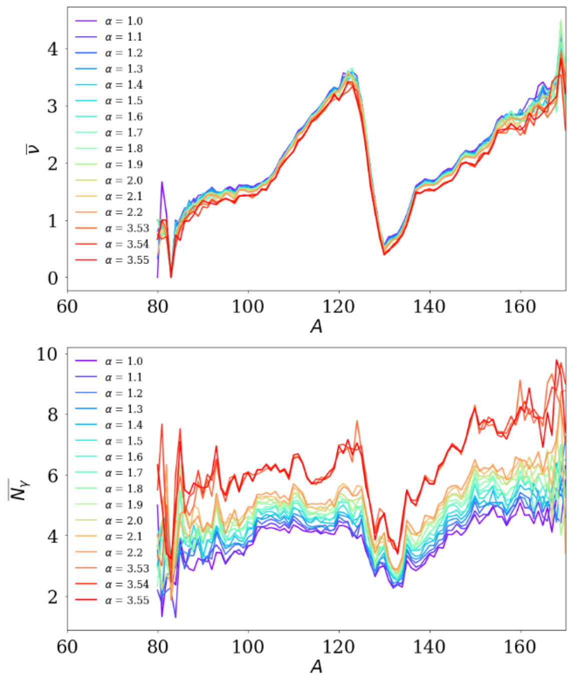

Fig. 10. Neutron and Gamma multiplicities as functions of fragment mass with various α

Both the average neutron and gamma multiplicities have strong linear correlations with α, as shown in Fig. 11.

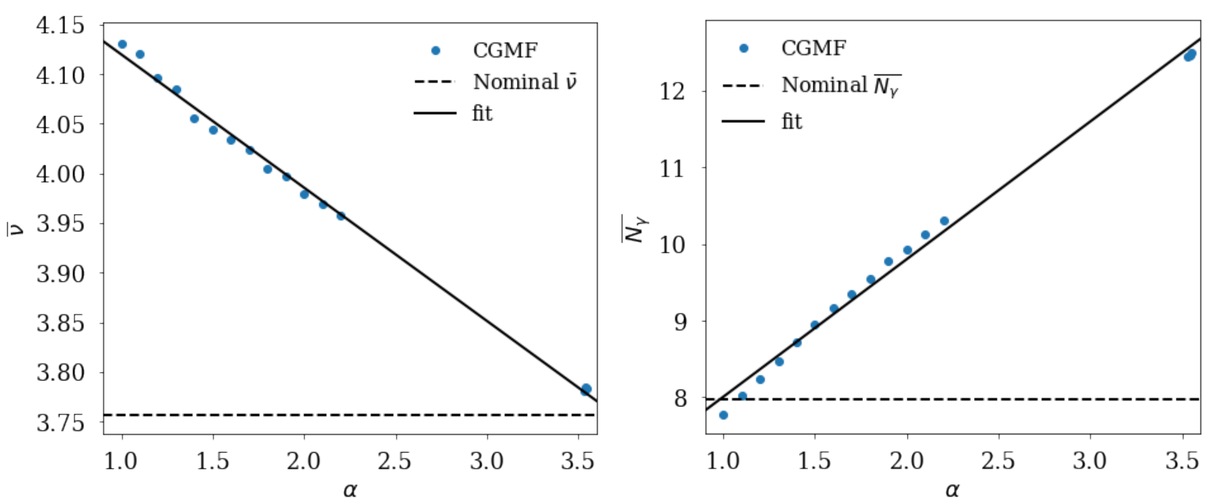

Fig. 11. Sensitivity of $\bar{\nu}$ and $\overline{N}_\gamma$ to α

8. Conclusions and Future Work

A Monte Carlo based fission event generator, CGMF, was developed and is undergoing analysis of the input parameters such that important fission observables can be consistently reproduced. This paper discusses the effects of numerous input parameters to various γ ray and neutron observables. The observables produced by CGMF are generated to account for phase space correlations, which is important for advanced and detailed simulations of fission events.

A new model for sampling Y(A,TKE) was implemented into CGMF based on modes or "fission channels," as presented by Brosa et al. (Ref. 3). This method for Y(A,TKE) sampling eliminates CGMF's dependence on yield data being read from a file and presents a systematic way to sample these yields for nuclei that do not have yield data available. For Cf-252(sf), 3 modes and 17 parameters were incorporated in the Brosa Modes Y(A,TKE). An initial optimization was performed to obtain nominal parameter values, and a sensitivity analysis was performed to see the effects of the 17 parameters on 20 neutron and γ ray responses. This sensitivity analysis made it possible to put constraints on the parameters, using experimental uncertainty bounds for $\bar{\nu}$ and $\langle TKE \rangle$. The parameters with the most significant sensitivities ($d_{max}^{S2}, \bar{A}_h^{S2}, d_{max}^{S1},$ and w_{S2}) can be tightly constrained according to this analysis.

Additional sensitivity analyses were performed with parameters R_T and α for Cf-252(sf). The R_T parameter controls the sharing of excitation energies between fission fragments and has a 2nd order effect on $\bar{\nu}$. An unrealistic value of R_T was found to produce $\bar{\nu}$, and the author suggests tuning the TKE of the fragments before R_T to produce and accurate $\bar{\nu}$. The α parameter affects the spin distribution of the fragments, in that a higher value of alpha produces a larger spin, which then inhibits neutron emission. These effects were seen with the linear correlations shown in the sensitivities of $\bar{\nu}$ and $\bar{N}_\gamma$ to α.

With the calculated sensitivity coefficients, optimization of the CGMF input parameters can be performed to match as many experimental data as possible for Cf-252(sf). The author recommends a Unified Monte Carlo (UMC) (Ref. 6) scheme be used for parameter evaluation. The work of expanding similar sensitivity analyses and optimizations to other compound nuclei and neutron induced reactions is also necessary for the applications of CGMF.

Acknowledgements

This work was performed at Los Alamos National Laboratory, under the auspices of the National Nuclear Security Administration of the US Department of Energy at Los Alamos National Laboratory under Contract No. DE-AC52-06NA25396. This work was partly supported by the Office of Defense Nuclear Nonproliferation Research & Development (DNN R&D), National Nuclear Security Administration, US Department of Energy.

References

1. A. Göök, F.-J. Hambsch, and M. Vidali, *Prompt neutron multiplicity in correlation with fragments from spontaneous fission of ^{252}Cf*, Phys. Rev. C **90**, 064611 (2014).
2. A. D. Carlson, V. G. Pronyaev, D. L. Smith, Nancy M. Larson, Zhenpeng Chen, G. M. Hale, F-J. Hambsch et al. *International evaluation of neutron cross section standards*, Nuclear Data Sheets **110**, no. 12: 3215-3324 (2009).
3. U. Brosa, H.-H. Knitter, *Proc. of the 18th International Symposium on Nuclear Physics*, Gaußig 1988. Org. by Technical University Dresden, Seeliger, D., Unholzer, S. (eds.), Rossendorf, Zentralinstitut für Kernforschung (1990).
4. E. Jones, E. Oliphant, P. Peterson, *et al. SciPy: Open Source Scientific Tools for Python*, 2001, http://www.scipy.org/ [Online; accessed 2017-08-17].
5. A. C. Wahl, *Systematics of fission-product yields*. No. LA-13928. Los Alamos Nat. Lab., (2002).
6. R. Capote, and D. L. Smith, *An Investigation of the Performance of the Unified Monte Carlo Method of Neutron Cross Section Data Evaluation,* Nuclear Data Sheets **109,** 2768-2773 (2008).

Appendix A: Example "histories-vectors.CGMF" file

```
# 98252 0 30000 1.7
119 46 22.398 1.5 1 105.368 3 4 0
896.644 4329.176 -1949.049 912.001 4240.267 -1869.834
0.017 0.007 0.053 1.491 -0.042 0.062 -0.030 3.062 -0.030 0.013 0.005 0.521 -0.003 0.008 -0.017 0.170 -
0.001 0.022 -0.023 0.464 0.029 0.025 -0.037 1.341
0.968 -1.156 -0.779 1.697 0.944 -1.128 -0.760 1.656 0.036 -0.167 -0.637 0.659 0.036 -0.167 -0.636 0.659
-1.077 -0.752 0.579 1.436 -1.114 -0.777 0.599 1.484 0.112 0.175 0.270 0.340 0.110 0.173 0.266 0.336
133 52 8.343 11.5 -1 94.279 0 6 0
-896.644 -4329.176 1949.049 -896.644 -4329.176 1949.049
-0.893 2.245 3.817 4.517 -0.901 2.264 3.849 4.556 -0.706 -1.004 0.914 1.530 -0.719 -1.023 0.931 1.559
1.326 0.859 -0.026 1.580 1.284 0.832 -0.025 1.530 0.607 -0.112 0.020 0.617 0.584 -0.108 0.019 0.594
0.032 0.028 -0.013 0.045 0.031 0.027 -0.013 0.043 -0.010 0.012 0.051 0.053 -0.010 0.012 0.051 0.054
112 44 22.140 14.0 1 105.441 3 5 0
-4273.729 1591.524 -1089.529 -4161.887 1553.021 -1046.095
0.033 0.030 0.010 1.001 -0.068 0.015 -0.048 3.364 -0.027 0.016 -0.017 0.590 -0.007 0.020 -0.019 0.394 -
0.007 -0.030 0.010 0.487 -0.045 0.006 0.021 1.143
-0.272 0.331 -0.204 0.474 -0.279 0.340 -0.209 0.487 0.021 -0.134 0.113 0.177 0.020 -0.134 0.113 0.176
0.005 0.063 0.038 0.074 0.005 0.063 0.038 0.074 -0.050 -0.065 -0.055 0.098 -0.051 -0.066 -0.056 0.101
0.112 0.010 -0.069 0.132 0.108 0.010 -0.067 0.127
140 54 12.730 8.0 1 84.339 2 2 0
4273.729 -1591.524 1089.529 4161.960 -1561.321 1078.621
0.049 0.006 -0.031 1.609 0.068 -0.058 0.014 3.888 0.001 0.012 0.042 0.893 0.051 0.026 -0.003 1.535
0.154 -0.448 -0.095 0.484 0.153 -0.444 -0.094 0.479 0.281 0.174 0.488 0.589 0.276 0.171 0.480 0.579
103 40 18.046 16.5 -1 101.700 2 8 0
-594.820 -2098.206 3839.903 -518.831 -2053.371 3795.438
0.025 0.011 -0.041 1.161 -0.052 -0.032 0.057 3.250 -0.042 0.025 -0.024 1.379 -0.029 -0.016 -0.009 0.563
1.378 -0.804 0.744 1.760 1.329 -0.775 0.718 1.698 1.237 0.092 0.074 1.242 1.181 0.088 0.071 1.186 -
0.190 0.301 0.215 0.416 -0.194 0.308 0.220 0.425 -0.206 0.094 0.096 0.246 -0.214 0.098 0.100 0.256 -
0.083 0.118 0.403 0.428 -0.084 0.119 0.407 0.432 -0.058 -0.229 -0.183 0.299 -0.059 -0.231 -0.185 0.302 -
0.134 0.156 -0.085 0.223 -0.138 0.161 -0.088 0.229 0.040 -0.050 0.074 0.098 0.040 -0.049 0.073 0.096
149 58 20.498 13.5 1 70.287 3 3 0
594.820 2098.206 -3839.903 533.811 2094.793 -3784.843
-0.033 0.051 -0.010 1.798 -0.002 -0.046 -0.024 1.284 -0.013 -0.043 0.001 0.935 0.012 0.045 0.005 1.027
0.012 0.002 -0.052 1.350 0.056 0.005 -0.039 2.197
0.368 0.191 -0.285 0.503 0.360 0.187 -0.278 0.491 0.363 -0.128 -0.140 0.410 0.353 -0.125 -0.136 0.398 -
0.203 -0.017 -0.159 0.258 -0.208 -0.018 -0.163 0.265
```

Appendix B: Example CGMF Observables Summary File

```
#CGMF SUMMARY FILE
#Generated: 2017-08-08 12:49
#Number of CGMF Output Files:  8
#
#File                                # Hist  w0      <TXE>    1sig        <TKE>   1sig         <J>      1sig
#-----------------------------------------------------------------------------------------------------------------
histories-vectors.CGMF.252CfYIELDS0     64000   0.813    33.333   1.067E+01    184.814 1.120E+01    10.819   6.081E+00
histories-vectors.CGMF.252CfYIELDS1     64000   0.600    37.987   1.392E+01    181.051 1.363E+01    11.071   6.243E+00
histories-vectors.CGMF.252CfYIELDS6     64000   0.722    35.273   1.233E+01    183.285 1.238E+01    10.918   6.139E+00
histories-vectors.CGMF.252CfYIELDS7     64000   0.747    34.749   1.196E+01    183.671 1.217E+01    10.901   6.119E+00
histories-vectors.CGMF.252CfYIELDS8     64000   0.771    34.217   1.151E+01    184.075 1.182E+01    10.886   6.121E+00
histories-vectors.CGMF.252CfYIELDS9     64000   0.796    33.657   1.105E+01    184.603 1.143E+01    10.834   6.092E+00
histories-vectors.CGMF.252CfYIELDS10    64000   0.820    33.112   1.051E+01    184.922 1.109E+01    10.813   6.069E+00
```

```
<nn>     n-mom2  n-mom3  <ng>     g-mom2  g-mom3  <En>CM  1sig          <Eg>CM  1sig          <En>Lab 1sig          <Eg>Lab 1sig
--------------------------------------------------------------------------------------------------------------------------------
3.878    13.136  38.219  9.374    92.922  971.10  1.890   1.586E+00     0.902   8.729E-01     2.119   1.633E+00     0.735   8.325E-01
4.289    16.558  56.161  9.544    96.301  1022.92 2.023   1.703E+00     1.004   9.952E-01     2.230   1.746E+00     0.809   9.389E-01
4.051    14.578  45.776  9.401    93.415  976.22  1.953   1.641E+00     0.946   9.283E-01     2.168   1.684E+00     0.767   8.802E-01
4.004    14.204  43.844  9.407    93.446  976.07  1.944   1.641E+00     0.931   9.092E-01     2.163   1.684E+00     0.756   8.639E-01
3.960    13.820  41.822  9.423    93.889  984.55  1.920   1.618E+00     0.918   8.943E-01     2.141   1.659E+00     0.746   8.496E-01
3.909    13.402  39.625  9.359    92.586  964.26  1.903   1.601E+00     0.907   8.819E-01     2.128   1.645E+00     0.739   8.405E-01
3.862    13.009  37.582  9.339    92.159  956.44  1.883   1.582E+00     0.896   8.662E-01     2.112   1.625E+00     0.731   8.265E-01
```

Appendix C: Example BrosaYields Class Input File

```
// Cf252 Brosa Mode data:
98252
w 0.8128359323532158 0.1734091083059585 0.013754959340825731
dmin 14.4946848 10.8924571 16.7133975
dmax 18.2042691 17.322745 20.4274353
ddec 0.446051597 0.134703577 0.165676002
Abar 144.542941 136.328441 126.0
sigA 6.11049848 3.69590546 18.225735
```

Appendix D: BrosaYields Class Source Code

```
/*
 *  CGMF
 *  Version: 1.0.6
 *
 *  [ FissionFragments.h ]
 *
 *  Generates primary fission fragment yields to be sampled in CGMF. It either
 *  reads data from input file, or produce them from systematics.
 *
 */

#ifndef __BROSAYIELDS_H__
```

```
#define __BROSAYIELDS_H__
#endif

// i'm using std::vectors
#include <vector>
#include <string.h>

using namespace std;

class BrosaYields
{
	/* PRIVATE */
private:
	// The data that is important for BrosaYields
	// the parameters that are necessary for the yield constructions
	// these will be "push_back(ed)" in the constructor


	/* PUBLIC */
public:

	vector<double> w;
	vector<double> Abar;
	vector<double> sigA;
	vector<double> dmin;
	vector<double> dmax;
	vector<double> ddec;

	int ZAIDc, Ac, Zc; // Original (before any pre-fission neutron emission) compound fissioning
nucleus

	//   7/26/17 values - Cf 252
	static const int NUMMODE = 3;
	static const int NUMA= 300;
	static const  int NUMTKE = 300;

	static const  int    NUMdZ  =   21; // [-dZ:+dZ] if dZ=10 for charge distribution around most
probable Zp[A]
	static const  int    NUMZ=  100; // number of charges Z


	// vectors of A and TKE
	// initialized in constructor
	int A[NUMA];
	int TKE[NUMTKE];
	int Amin;
	int Amax;
	int TKEmin;
```

```
	int TKEmax;
	int Zmin, Zmax;
	int Zt;
	double PE;

	// Wahl's parameters

	int dZ;
	double Z0[NUMA];
	double sZ0[NUMA];
	double FZZ0[NUMA];
	double FNZ0[NUMA];

	double sigmaZ;

	double YA[NUMA];
	double YZA[NUMdZ][NUMA];
	double YZA2[NUMZ][NUMA];


	// array of TKE|A for each mode
	// first index is mode number
	// second index is A (row)
	// third index is probability of TKE|A
	double YTKEA[NUMMODE][NUMA][NUMTKE];

	static double e2(){
		return 1.4399643929;// MeV fm
	}

// Constructor: when implemented, a file is read with the necessary parameters for the Brosa modes
	// necessary data in input file: h (equivalently the weight w), Abar, sigA, dmax, dmin, ddec
	// for each mode
	BrosaYields (string inputFilename, int ZAID, int A_min);

	~BrosaYields (void);

	// function to read file for input parameters
	void readfile(string inputFilename);

	// function to build YTKEA
	void buildYTKEA();



	void buildYA();

	// A method that would be necessary would be finding out what mode we are dealing with
```

```
	int samplemode(); // returns a mode number according to the order of that in the input file
	// I intend for the order to be S2, S1, SL

	// let's have a public method that can sample an A
	// ALWAYS SAMPLES THE HEAVY FRAGMENT
	int sampleA(int i_mode);

	// And another method that samples Y(TKE|A)
	int sampleTKEA(int A, int mode);

	void computeWahlParameters();

	void buildYZA();

	int sampleZA(int A);

};


/*
 *  CGMF
 *  Version: 1.0.7.2
 *  Austin Carter
 *
 *  [ BrosaYields.cpp ]
 *
 */

// some libraries
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <cmath>
#include <iomanip>
#include <sstream>
#include <stdio.h>
#include <string.h>


#include "config.h"
//for the random numbers
#include "mt19937ar.h"
// has constants: pi, etc.
#include "physics.h"

// our namespace
using namespace std;
```

```
#include "BrosaYields.h"

// CONSTRUCTOR
BrosaYields::BrosaYields(string filein, int ZAID, int A_min){

  // initialize stuff
  Amin = A_min;
  Amax = NUMA - 1;
  TKEmin = 100;
  TKEmax = NUMTKE - 1;
  Zt = Zc;
  PE = 0.0;
  dZ = 3;
  PE = 0.0;



	ZAIDc = ZAID;
	Zc=int(ZAIDc/1000.0);
	Ac=ZAIDc-1000*Zc;
	Zt = Zc;
	PE = 0.0;

	// populate A and TKE vectors
	for (int i = 0; i < NUMA; i++){
		A[i] = i;
	}

	for (int i = 0; i < NUMTKE; i++){
		TKE[i] = i;
	}

	// put parameter data into vectors from file
	readfile(filein);
  //cout << "readfile good" << endl;

	// build the YA array
	buildYA();
  //cout << "buildYA good" << endl;
	// build the Y(TKE|A)_m array
	buildYTKEA();
  //cout << "buildYTKEA good" << endl;

	// build the Y(Z|A) array
	computeWahlParameters();
  //cout << "computeWahlParameters good" << endl;
	buildYZA();
```

```
  //cout << "buildYZA good" << endl;




}

// Destructor
BrosaYields::~BrosaYields () {}

void BrosaYields::readfile(string inputFilename){

	string f = DATADIR;
	f += inputFilename;
	ifstream brosadata (&f[0],ios::in);

	if (!brosadata) {
	cerr << "[BrosaYields::readfile(inputFilename)] Brosa Yields parameters data file could not be found" << f <<  endl;
	exit(-1);
	}

	string line;
	string strtmp;
	double dubtmp;

	while(getline(brosadata,line)){
		stringstream ss(line);
		//found the w
		if(int(line.find("w")) == 0){
			ss >> strtmp;
			for (int i = 0; i < NUMMODE; i++){ss >> dubtmp; w.push_back(dubtmp);}
		}
		if(int(line.find("dmin")) == 0){
			ss >> strtmp;
			for (int i = 0; i < NUMMODE; i++){ss >> dubtmp; dmin.push_back(dubtmp);}
		}
		if(int(line.find("dmax")) == 0){
			ss >> strtmp;
			for (int i = 0; i < NUMMODE; i++){ss >> dubtmp; dmax.push_back(dubtmp);}
		}
		if(int(line.find("ddec")) == 0){
			ss >> strtmp;
			for (int i = 0; i < NUMMODE; i++){ss >> dubtmp; ddec.push_back(dubtmp);}
		}
		if(int(line.find("Abar")) == 0){
			ss >> strtmp;
			for (int i = 0; i < NUMMODE; i++){ss >> dubtmp; Abar.push_back(dubtmp);}
```

```
			}
			if(int(line.find("sigA")) == 0){
				ss >> strtmp;
				for (int i = 0; i < NUMMODE; i++){ss >> dubtmp; sigA.push_back(dubtmp);}
			}
		}

}

void BrosaYields::buildYA(){
  std::fill_n (YA, NUMA, 0.0);
		for(int i = Amin; i < NUMA; i++){
			YA[i] = 0.0;
			for (int j = 0; j < 3; j++){
				YA[i] += 1/sqrt(8* pi * sigA[j]*sigA[j])*
				(exp(-(A[i] - Abar[j])*(A[i] - Abar[j])/2./sigA[j]/sigA[j]) +
					exp(-(A[i] - Ac +Abar[j]) * (A[i] - Ac +Abar[j]) / 2. / sigA[j] / sigA[j]));
			}
		}
}

void BrosaYields::buildYTKEA(){
		// okay, the way this large array is structured is as follows:
		// 3 modes that act as pages; the first index element
		// then for the 2d array (contained within each mode):
		//     TKE ---------->
		// A
		// |
		// |
		// |
		// ^
		// where the distributions are normalized to 1.0 along the row which are sampled in
sampleTKEA

		double T;
		double TKEAtotal;

		for (int i = 0; i < NUMMODE; i++){
			for (int j = 0; j < NUMA; j++){
    // 0 the array
    for (int k = 0; k < NUMTKE; k++){
      YTKEA[i][j][k]= 0.0;
    }
				TKEAtotal = 0.;
				for (int k = TKEmin; k < NUMTKE; k++){
					T = ((double)Zc/(double)Ac)*((double)Zc/(double)Ac) * ((double)Ac -
(double)A[j])*(double)A[j] * e2() / (double)TKE[k] - dmin[i];
					if(T > 0.){
```

```
                                        YTKEA[i][j][k] = (200./(double)TKE[k])*(200./(double)TKE[k])*
exp(2. * (dmax[i] - dmin[i])/ ddec[i] -
                                                T/ddec[i] - (dmax[i] - dmin[i])*(dmax[i] - dmin[i]) /T
/ddec[i])
                                                * (double)YA[j];
                                } else {
                                        YTKEA[i][j][k] = 0.;
                                }
                                TKEAtotal += YTKEA[i][j][k];

                        }

                        // avoid the nans
                        if (TKEAtotal == 0.0){
                                TKEAtotal = 0.01;
                        }
                        // normalize to 1.0 for each row
                        for (int k = TKEmin; k < NUMTKE; k++){
                                YTKEA[i][j][k]= YTKEA[i][j][k] / TKEAtotal;
                        }
                }
        }

}

int BrosaYields::samplemode(){
        // get a random number
        double r1 = genrand_real3();

        // sample probability mass dist for mode
        double wtot = 0.;
        int i_mode;
        for(int i = 0; i < NUMMODE; i++){
                wtot += w[i];
                if (r1 <= wtot){
                        i_mode = i;
                        break;
                }
        }
        return i_mode;

}


int BrosaYields::sampleA(int i_mode){

        // sample the gaussian to find A for the HEAVY Fragment
        double r2 = genrand_real3();
```

```
        double r3 = genrand_real3();

        double A_s = Abar[i_mode] + sigA[i_mode] * sqrt(-2. * log(r2)) * cos(2. * pi * r3);


        // ALWAYS SAMPLES THE HEAVY FRAGMENT

        // return the heavy fragment as rounded integer
        return int(A_s+0.5);
}

int BrosaYields::sampleTKEA(int A, int i_mode){

        // we use YTKEA as a probability mass distribution
        // find the index of A
        int i_A = (int)(A);

        double r4 = genrand_real3();

        double KEtot = 0.0;

        for (int i_TKE = 0; i_TKE < NUMTKE; i_TKE++){
                KEtot += YTKEA[i_mode][i_A][i_TKE];
                if (r4 <= KEtot){
    //cout << "sampled TKE|A" << endl;
                        return (int)(i_TKE);
                }
        }

        // should not get here
        cerr << "ERROR: Could not sample KE distribution in BrosaModes class." << endl;
  exit(-1);
}

int BrosaYields::sampleZA(int A){
        int i_A = int(A);
        double r5 = genrand_real3();

        // normalize the column vector in YZA (because we sample Z based on A)
        double YZA_tot = 0.0;
        for (int i = 0; i < NUMZ; i++){
                YZA_tot += YZA2[i][i_A];
        }
        //create a new vector to be sampled from
        double YZA_temp[NUMZ];
        for (int i = 0; i<NUMZ; i++){
                YZA_temp[i] = YZA2[i][i_A] / YZA_tot;
        }
```

```
		// sample the probability mass distribution
		double Ztot = 0.0;
		for (int i_Z = 0; i_Z < NUMZ; i_Z++){
			Ztot += YZA_temp[i_Z];
			if(r5 <= Ztot){
				return i_Z;
			}
		}
		// should never get here
		cerr << "ERROR: Could not sample Z distribution in BrosaModes class." << endl;
   exit(-1);
}

void BrosaYields::buildYZA(){
		double sigma, c=0.0, Zp0;
  int iZp0, iZ;
  int i, j;
  double oddeven=1.0, V, W;

  sigma = 0.4;
  //  c=2.0*(sigma*sigma+1.0/12.0);

  for (i=Amin; i<=Amax; i++) {
    c = sZ0[i];
    Zp0 = Z0[i];
    iZp0 = (int) floor(Zp0+0.5);

    for (j=0; j<=2*dZ; j++) {

      //      Zi = Zp0+j-dZ;
      iZ = iZp0+j-dZ;
      YZA[j][i] = 0.0;
      YZA2[iZ][i] = 0.0;

      if (c>0.0) {
        //       Zi = double(int(Zp0+0.5)+j);
        if (iZ%2==0) {
          if (i%2==0) {
            oddeven = FZZ0[i]*FNZ0[i]; // even Z - even N
          } else {
            oddeven = FZZ0[i]/FNZ0[i]; // even Z - odd N
          }
        } else {
          if ((i+1)%2==0) {
            oddeven = FNZ0[i]/FZZ0[i]; // odd Z - even N
          } else {
            oddeven = 1.0/(FZZ0[i]*FNZ0[i]); // odd Z - odd N
```

```
      }
    }

    V = (iZ-Zp0+0.5)/(sqrt(2.0)*c);
    W = (iZ-Zp0-0.5)/(sqrt(2.0)*c);

    YZA[j][i] = oddeven * 0.5 * ( erf(V) - erf(W) ); // YA[i]
    YZA2[iZ][i] = YZA[j][i];

    //   ExpYields%Yza(j,i) = oddeven * 1.0_rk/sqrt(c*pi)*exp(-1.0_rk* ( Zi-Zp )**2/c) * 1.0_rk
!ExpYields%Ya(i)

   } else {

    if (j==dZ) {
     //         YZA[j][i] = 1.0; // yields%Ya(i)
     //              YZA2[iZ][i] = 1.0;
     YZA[j][i] = YA[i];
     YZA2[iZ][i] = YA[i];
    }
   }

  } // end loop over Z


  // TEMPORARY
=================================================================================================
==================
  //                  exit(0);
  // TEMPORARY
=================================================================================================
==================

  double x=0;
  for (j=0; j<=2*dZ; j++) { x += YZA[j][i]; }
  if (x!=0) {
   for (j=0; j<=2*dZ; j++) {
    YZA[j][i] = YZA[j][i] * YA[i] / x;
   }
  }

  double y=0;
  for (j=0; j<NUMZ; j++) { y += YZA2[j][i]; }
  if (y!=0) {
   for (j=0; j<NUMZ; j++) {
    YZA2[j][i] = YZA2[j][i] * YA[i] / y;
   }
  }
```

```
  } // end loop over A

  /*  ofstream out;
   out.open("yza");
   for (i=Amin; i<=Amax; i++) {
   out << "\n#\n\n";
   int z = int(Z0[i]+0.5)-dZ;
   for (j=0; j<=2*dZ; j++) {
                                  out << i << " " << z+j << " " << YZA[j][i] << "\n";
   }
   }
   out.close();
   */


}

void BrosaYields::computeWahlParameters(){
            double sigz140t, delz140t, Fz140t, Fn140t, sigzSLt, delzSLt, FzSLt, SL50t,
  sigz50t, delzmaxt, sigzSLWt, delzSLWt, FzSLWt, FnSLWt;

  double delz[NUMA], sigz[NUMA], Fz[NUMA], Fn[NUMA];

  double sigz140[5] = {0.566, 0.0, 0.0064, 0.0109, 0.0};
  double delz140[5] = {-0.487, 0.0, 0.0180, 0.0, -0.00203};
  //  double Fz140[5]   = {1.207, 0.0, -0.0420, 0.0, 0.0022};
  double Fz140[5]   = {1.242, 0.0, -0.0183, -0.0152, 0.0};
  double Fn140[5]   = {1.076, 0.0, 0.0, 0.0, 0.0};
  double sigzSL[5]  = {-0.0038, 0.0, 0.0, 0.0, 0.0};
  double delzSL[5]  = {-0.0080, 0.0, 0.0, 0.0, 0.0};
  double FzSL[5]    = {0.0030, 0.0, 0.0, 0.0, 0.0};
  double SL50[5]    = {0.191, 0.0, -0.0076, 0.0, 0.0};
  double sigz50[5]  = {0.356, 0.060, 0.0, 0.0, 0.0};
  double delzmax[5] = {0.699, 0.0, 0.0, 0.0, 0.0};
  double sigzSLW[5] = {-.045, 0.0094, 0.0, 0.0, 0.0};
  double delzSLW[5] = {0.0, -0.0045, 0.0, 0.0, 0.0};
  double FzSLW[5]   = {0.159, -0.028, 0.0, 0.0, 0.0};
  double FnSLW[5]   = {0.039, 0.0, 0.0, 0.0, 0.0};

  std::fill_n(Z0,NUMA,0.0);
  std::fill_n(sZ0,NUMA,0.0);
  std::fill_n(FZZ0,NUMA,0.0);
  std::fill_n(FNZ0,NUMA,0.0);

  //============================================================================
  // TEMPORARY... FOR ANYTHING BUT n+U-235 FISSION
  //============================================================================
```

```
/*  if (Ac!=236) {

 for (int i=Amin; i<=Amax; i++) {
 delz[i]  = 0.5; // dZ
 if(i>Asym) delz[i] = -0.5; // dZ
 sigz[i] = sigmaZ; // width of distribution
 Fz[i] = 1.0; // FZ - odd-even factor
 Fn[i] = 1.0; // FN - odd-even factor
 }

 //-- save Wahl parameters
 double x = float(Zt)/Ac;
 for (int i=Amin; i<=Amax; i++) {
 Z0[i]   = (x*i+delz[i]);
 sZ0[i]  = sigz[i];
 FZZ0[i] = Fz[i];
 FNZ0[i] = Fn[i];
 //      cout << i << " " << Z0[i] << " " << sZ0[i] << "\n";
 }

 //-- Find true (Zmin, Zmax)
 Zmin = (int) floor(Z0[Amin]+0.5)-dZ;
 Zmax = (int) floor(Z0[Amax]+0.5)+dZ;

 return;

 }
 */


// Calculate input parameter for any given fissioning system
// >> SHOULD 6.551 BE REPLACED BY neutronSeparationEnergy? I THINK SO...
// WHAT ABOUT (92,236)... IS IT JUST VALID FOR U235?
if (PE <= 8.0) {

  sigz140t = sigz140[0]+sigz140[1]*(Zt-92)+sigz140[2]*(Ac-236)+sigz140[3]*(PE-6.551)+sigz140[4]*(Ac-
236)*(Ac-236);
  delz140t = delz140[0]+delz140[1]*(Zt-92)+delz140[2]*(Ac-236)+delz140[3]*(PE-
6.551)+delz140[4]*(Ac-236)*(Ac-236);
  Fz140t   = Fz140[0]+Fz140[1]*(Zt-92)+Fz140[2]*(Ac-236)+Fz140[3]*(PE-6.551)+Fz140[4]*(Ac-236)*(Ac-
236);
  Fn140t   = Fn140[0]+Fn140[1]*(Zt-92)+Fn140[2]*(Ac-236)+Fn140[3]*(PE-6.551)+Fn140[4]*(Ac-
236)*(Ac-236);
  sigzSLt  = sigzSL[0]+sigzSL[1]*(Zt-92)+sigzSL[2]*(Ac-236)+sigzSL[3]*(PE-6.551)+sigzSL[4]*(Ac-236)*(Ac-
236);
  delzSLt  = delzSL[0]+delzSL[1]*(Zt-92)+delzSL[2]*(Ac-236)+delzSL[3]*(PE-6.551)+delzSL[4]*(Ac-
236)*(Ac-236);
  FzSLt    = FzSL[0]+FzSL[1]*(Zt-92)+FzSL[2]*(Ac-236)+FzSL[3]*(PE-6.551)+FzSL[4]*(Ac-236)*(Ac-236);
```

```
   SL50t    = SL50[0]+SL50[1]*(Zt-92)+SL50[2]*(Ac-236)+SL50[3]*(PE-6.551)+SL50[4]*(Ac-236)*(Ac-236);
   sigz50t  = sigz50[0]+sigz50[1]*(Zt-92)+sigz50[2]*(Ac-236)+sigz50[3]*(PE-6.551)+sigz50[4]*(Ac-
236)*(Ac-236);
   delzmaxt = delzmax[0]+delzmax[1]*(Zt-92)+delzmax[2]*(Ac-236)+delzmax[3]*(PE-
6.551)+delzmax[4]*(Ac-236)*(Ac-236);
   sigzSLWt = sigzSLW[0]+sigzSLW[1]*(Zt-92)+sigzSLW[2]*(Ac-236)+sigzSLW[3]*(PE-
6.551)+sigzSLW[4]*(Ac-236)*(Ac-236);
   delzSLWt = delzSLW[0]+delzSLW[1]*(Zt-92)+delzSLW[2]*(Ac-236)+delzSLW[3]*(PE-
6.551)+delzSLW[4]*(Ac-236)*(Ac-236);
   FzSLWt   = FzSLW[0]+FzSLW[1]*(Zt-92)+FzSLW[2]*(Ac-236)+FzSLW[3]*(PE-6.551)+FzSLW[4]*(Ac-
236)*(Ac-236);
   FnSLWt   = FnSLW[0]+FnSLW[1]*(Zt-92)+FnSLW[2]*(Ac-236)+FnSLW[3]*(PE-6.551)+FnSLW[4]*(Ac-
236)*(Ac-236);

  } else if (PE<=20.0) {

   sigz140[0] = 0.542; sigz140[1] = 1.310; sigz140[2] = 0.033; sigz140[3] = 0.0; sigz140[4] = -0.005;
   delz140[0] = -0.428; delz140[1] = 0.0; delz140[2] = 0.0; delz140[3] = 0.164; delz140[4] = -0.0116;
   SL50[0] = 0.191; SL50[1] = 0.0; SL50[2] = -0.0076; SL50[3] = 0.0; SL50[4] = 0.0;
   sigz50[0] = 0.542; sigz50[1] = 1.310; sigz50[2] = 0.033; sigz50[3] = 0.0; sigz50[4] = -0.005;

   sigz140t = (sigz140[0]+sigz140[2]*(Zt-92))+((sigz140[1]+sigz140[3]*(Zt-92))-(sigz140[0]+sigz140[2]*(Zt-
92)))*(1.0-exp(-sigz140[4]*PE));
   delz140t = (delz140[0]+delz140[2]*(Zt-92))+((delz140[1]+delz140[3]*(Zt-92))-
(delz140[0]+delz140[2]*(Zt-92)))*(1.0-exp(-delz140[4]*PE));
   Fz140t   = 1.0;
   Fn140t   = 1.0;
   sigzSLt  = 0.0;
   delzSLt  = 0.0;
   FzSLt    = 0.0;
   SL50t    = (SL50[0]+SL50[2]*(Zt-92))+((SL50[1]+SL50[3]*(Zt-92))-(SL50[0]+SL50[2]*(Zt-92)))*(1.0-exp(-
SL50[4]*PE));
   sigz50t  = (sigz50[0]+sigz50[2]*(Zt-92))+((sigz50[1]+sigz50[3]*(Zt-92))-(sigz50[0]+sigz50[2]*(Zt-
92)))*(1.0-exp(-sigz50[4]*PE));
   delzmaxt = 0.0;
   sigzSLWt = 0.0;
   delzSLWt = 0.0;
   FzSLWt   = 0.0;
   FnSLWt   = 0.0;


  } else {

   cerr << "[computeWahlParameters] Wahl parameters only defined up to Einc+Sn=20 MeV!" << endl;
   exit(-1);

  }
```

```
// Calculate the region values

double F1=floor((250.-Ac)/14.+0.5);  // F1=anint((250.-Ac)/14.); in F95
double F2=1.-F1;
double AK1=50.0*float(Ac)/Zt-delzmaxt/SL50t;
double AK2=(50.0-delzmaxt)*float(Ac)/Zt;
double Apmax=F1*AK1+F2*AK2;

int B1=70;
int B2= (int) floor(77+0.036*(Ac-236)+0.5);      // nint() in F95
int B4= (int) floor((delzmaxt-delz140t+Apmax*SL50t+140*delzSLt)/(SL50t+delzSLt)+0.5); // nint() in F95
int B3=(Ac-B4);
int B5=(Ac-B2);
int B6=(Ac-B1);
int Bb=int(Apmax+0.5); // nint() in F95
int Ba=int(Ac-Apmax+0.5); // nint() in F95

// Calculate values of sigz, delz, Fn, Fz for give Ap values
for (int i=Amin; i<=Amax; i++) {
 // Peak Regions
 if ((i>=B2 && i<=B3) || (i>=B4 && i<=B5)) {
   if (i>Ac/2.0) {
     delz[i]=delz140t+delzSLt*(i-140.0);
     sigz[i]=sigz140t+sigzSLt*(i-140.0);
     Fz[i]=Fz140t+FzSLt*(i-140.0);
     Fn[i]=Fn140t+FzSLt*(i-140.0);
   } else if (i<Ac/2.0) {
     delz[i]=-1*delz140t+delzSLt*(i-(Ac-140));
     sigz[i]=sigz140t-sigzSLt*(i-(Ac-140));
     Fz[i]=Fz140t-FzSLt*(i-(Ac-140));
     Fn[i]=Fn140t-FzSLt*(i-(Ac-140));
   }
   // Fn(i)=Fn140t
 }
}

for (int i=Amin; i<=Amax; i++) {
 // Near Symmetry Region
 if (i>B3 && i<B4) {
   Fn[i]=1.;
   Fz[i]=1.;
   if (i>B3 && i<=Ba) {
     delz[i]=delz[B3]-SL50t*(i-B3);
     sigz[i]=sigz50t;
   } else if (i>Ba && i<Bb) {
     delz[i]=delz[Ba]+(i-Ba)*(2.0*delz[Ba])/(Ba-Bb);
     sigz[i]=sigz140t-sigzSLt*(140-Bb);
   } else if (i>=Bb && i<B4) {
```

```
    delz[i]=delz[B4]+SL50t*(B4-i);
    sigz[i]=sigz50t;
   }
  }
}

for (int i=Amin; i<=Amax; i++) {
  // Wing Regions
  if ((i>B1 && i<B2) || (i>B5 && i<B6)) {
   if (i>Ac/2.0) {
    delz[i]=delz[B5]-delzSLWt*(i-B5);
    sigz[i]=sigz[B5]+sigzSLWt*(i-B5);
    Fz[i]=Fz140t+FzSLWt*(i-B5);
    Fn[i]=Fn140t+FnSLWt*(i-B5);
   } else if (i<Ac/2.0) {
    delz[i]=delz[B2]+delzSLWt*(B2-i);
    sigz[i]=sigz[B5]+sigzSLWt*(B2-i);
    Fz[i]=Fz140t+FzSLWt*(B2-i);
    Fn[i]=Fn140t+FnSLWt*(B2-i);
   }
  }

  // Far Wing Regions
  if (i<=B1 || i>=B6) {
   if (i>Ac/2.0) {
    delz[i]=delz[B5];
    sigz[i]=sigz[B5];
    Fz[i]=Fz140t;
    Fn[i]=Fn140t;
   } else if (i<Ac/2.0) {
    delz[i]=delz[B2];
    sigz[i]=sigz[B5];
    Fz[i]=Fz140t;
    Fn[i]=Fn140t;
   }
  }
}

//-- save Wahl parameters
double x = float(Zt)/Ac;
for (int i=Amin; i<=Amax; i++) {
  Z0[i]   = (x*i+delz[i]);
  sZ0[i]  = sigz[i];
  FZZ0[i] = Fz[i];
  FNZ0[i] = Fn[i];
  //    FZZ0[i] = 1.0;
  //    FNZ0[i] = 1.0;
  //    cout << i << " " << Z0[i] << " " << sZ0[i] << "\n";
```

```
  }

  //-- Find true (Zmin, Zmax)
  Zmin = (int) floor(Z0[Amin]+0.5)-dZ;
  Zmax = (int) floor(Z0[Amax]+0.5)+dZ;



}
```