CGMF User Manual

Release 1.1.0

Patrick Talou

Apr 07, 2022

Contents

CGMF is a code that simulates the emission of prompt fission neutrons and gamma rays from excited fission fragments right after scission. It implements a Monte Carlo version of the Hauser-Feshbach statistical theory of nuclear reactions to follow the decay of the fission fragments on an event-by-event basis. Probabilities for emitting neutrons and gamma rays are computed at each stage of the decay. Each fission event history records characteristics of the parent fragment (mass, charge, kinetic energy, momentum vector, excitation energy, spin, parity) and the number (multiplicity) and characteristics (energy, direction) of the prompt neutrons and gamma rays emitted in this event.

Recommended publication for citing

Patrick Talou, Ionel Stetcu, Patrick Jaffke, Michael E. Rising, Amy E. Lovell, and Toshihiko Kawano, "Fission Fragment Decay Simulations with the CGMF Code," to be submitted to Comp. Phys. Comm. (2020).

Support

For any questions related to **CGMF**, its use, and its code source, please email us at: **:email:'cgmf-help@lanl.gov <cgmf-help@lanl.gov>'**.

CHAPTER 1

Contents

1.1 Introduction

Note:

Current Version: 1.1
Date: Apr 07, 2022

1.1.1 Motivation and Physics Background

The fission of a heavy nucleus into two or more lighter fragments is usually accompanied with the emission of *prompt* neutrons and photons. In our current understanding of the fission process, the fission fragments are produced in a certain state of deformation and intrinsic excitation, eventually resulting in the production of excited fission fragments. Very quickly, those primary fragments will evaporate neutrons and photons to reach a more stable configuration, either a ground-state or a long-lived isomer. The post-neutron emission fission fragments, also called fission products, will possibly further β-decay, leading to another burst of β-delayed neutron and photon emissions.

The study of the prompt fission neutrons and photons is important to better model the nuclear fission process, constrain the collective and intrinsic configurations of the nascent fragments near the scission point, and understand the sharing of the available excitation energy between the two fragments. This study is also highly relevant for applications, ranging from nuclear energy safety and efficiency to non-proliferation and stockpile stewardship missions.

Until recently, most of the nuclear data evaluation work related to prompt fission neutrons and photons was limited to their average number, or multiplicity, and their average energy spectrum, as a function of incident neutron energy in the case of neutron-induced fission reactions. Even for those somewhat simple quantities, scarce experimental data exist only, limited to some important isotopes and incident neutron energies. Phenomenological models have been developed over the years, e.g., the so-called Los Alamos model (LAM), mostly for calculating the average prompt fission neutron spectrum (PFNS). Evaluated data on prompt fission always originated from very scarce experimental data, leaving important gaps even in the most modern nuclear data libraries such as ENDF/B-VIII.0.

The **CGMF** code was developed to model the de-excitation of the fission fragments on an event-by-event basis, following the successive emissions of neutrons and photons. This is radically different from what had been done in the past, allowing an unprecedented level of predictions on distributions and correlations of neutrons, photons and fission fragments. The development of this code is accompanied by a host of new fission experiments that look at increasing levels of details and correlations among the vast quantity of fission data. Correlations and distributions of post-scission data will be very useful to constrain the still free parameters entering the **CGMF** code and models.

1.1.2 Synopsis of the CGMF code

The **CGMF** code is based on two older codes developed at LANL: **FFD** [LA-CC-10-003] and **CGM** [LA-CC-11-018]. It performs Monte Carlo simulations of the decay of excited fission fragments by emission of prompt neutrons and gamma rays. The Hauser-Feshbach statistical theory of compound nuclear reactions is used to compute the emission probabilities at each step of the cascade. Monte Carlo histories are then recorded and analyzed.

The average prompt fission neutron multiplicity $\overline{\nu}$, the prompt fission neutron multiplicity distribution $P(\nu)$, the average prompt fission neutron multiplicity as a function of mass, charge and kinetic energy of the fragment, $\overline{\nu}(A, Z, TKE)$, etc, can all be extracted from **CGMF** calculations. Similar quantities can also be obtained for prompt gamma rays. In addition, $n-n$, $n-\gamma$, and $\gamma-\gamma$ correlations can be studied both in energy and angle.

The **FFD** code was the first fission fragment evaporation code to be developed, and used the Weisskopf-Ewing approximation to evaporate neutrons. Characteristics of the emitted prompt neutrons could be retrieved, but only little information could be inferred for the prompt γ-ray data, and no specific discrete transitions in particular fragments could be tagged. Meanwhile, the **CGM** code was being developed as a general Monte Carlo implementation of the traditional statistical nuclear reaction codes, e.g., GNASH, EMPIRE, TALYS, COH, using the very well-established Hauser-Feshbach statistical theory of nuclear reactions. With **CGM**, one can follow the decay of an excited compound nucleus by evaporation of photons, neutrons, and light charged particles until it reaches its ground-state or a long-lived isomer. Monte Carlo histories could be followed one-by-one to study correlations and *exclusive* data.

Initially written in FORTRAN 95, significant parts of **FFD** were re-written into C++ classes to work seamlessly with the C++ code **CGM**, leading to the release of the **CGMF** code that applies the physics of **CGM** to the de-excitation of fission fragments.

1.1.3 For more information

This online user manual is intended to become the main reference for **CGMF**. The main reference to cite **CGMF** is its official documentation published as **'Talou, Stetcu, Jaffke, Rising, Lovell and Kawano, submitted to Comp. Phys. Comm. (2020) <>'_**. In addition, several *Publications* and presentations might be of interest to the reader wanting more information on how the code is actually used for practical studies.

1.2 Getting Started

1.2.1 Obtaining CGMF

CGMF is an open source code and can be cloned from the github repository at https://github.com/lanl/CGMF by typing:

```
git clone git@github.com:lanl/CGMF.git
```

1.2.2 Installing CGMF

The build system for CGMF is based on the CMake build system generation tool. The current version requires CMake 3.16.2 minimumm and can be freely downloaded at https://cmake.org/download/. Assuming that the **CGMF** source tree is located in the $CGMFPATH directory, the Linux commands to build the code library and utilities are:

```
cd $CGMFPATH
mkdir build
cd build
cmake ..
make
```

This creates the static library *libcgmf.a* in the $CGMFPATH/build/libcgmf directory and also creates the executable *cgmf.x* in the $CGMFPATH/build/utils/cgmf directory. Other options for building the code include:

```
* CMAKE\_BUILD\_TYPE=(Debug, RelWithDebInfo, Release [default])
* CMAKE\_INSTALL\_PREFIX=(/usr/local/ [default])
* cgmf.shared\_library=(ON, OFF [default])
* cgmf.x.MPI=(ON, OFF [default])
```

1.2.3 Running CGMF

To launch a CGMF run, type:

```
./cgmf.x -i 98252 -e 0.0 -n 1000000
```

which means that CGMF is run for the spontaneous fission (incident energy is set to 0.0 (using *-e 0.0*) of 252Cf (ZAID=98252) with 1,000,000 fission events.

Table 1.1: CGMF Arguments

-i $ZAIDt	[required]	1000*Z+A of target nucleus, or fissioning nucleus if spontaneous fission
-e $Einc	[required]	incident neutron energy in MeV (0.0 for spontaneous fission)
-n $nevents	[required]	number of Monte Carlo fission events to run or to be read. If $nevents is negative, produces initial fission fragment yields Y(A,Z,KE,U,J,p)
-t $timeCoinc	[optional]	time coincidence window for long-lived isomer gamma-ray emission cutoff (in sec)
-d $datapath	[optional]	overrides the environment variable CGMFDATA and default datapath
-f $filename	[optional]	fission histories results file (“results.cgmf” is default)

The CGMF run above would create a history file (‘histories.cgmf’) as well as a concise summary of important average quantities on the console, such as:

```
//// CGMF Results ////

Reaction: spontaneous fission of (98,252)

Average Light Fragment (Z,A) = (42.56,108.40)
Average Heavy Fragment (Z,A) = (55.44,143.60)

Average Kinetic Energies: LF = 105.73 MeV ; HF = 80.07 MeV ; <TKE> = 185.79 MeV
```

(continues on next page)

(continued from previous page)

```
Average Excitation Energies: LF = 18.33 MeV ; HF = 13.61 MeV ; <TXE> = 31.94 MeV

Average Fragment Spins: <J>_LF = 9.11 hbar ; <J>_HF = 9.92 hbar ; <J> = 9.52 hbar

*** Prompt Fission Neutrons ***

Multiplicities (n/f):  <nu>_LF = 2.12 ; <nu>_HF = 1.69 ; <nu>_prefission = 0.00 ; <nu>
↪_tot = 3.82
c-o-m Energies:  <Ecm>_LF = 1.34 MeV ; <Ecm>_HF = 1.21 MeV ; <Ecm>_prefission = 0.00␣
↪MeV ; <Ecm>_tot = 1.28 MeV
Lab. Energies:   <Elab>_LF = 2.27 MeV ; <Elab>_HF = 1.72 MeV ; <Elab>_prefission = 0.
↪00 MeV ; <Elab>_tot = 2.02 MeV

*** Prompt Fission Gammas ***

Multiplicities (g/f):  <nu_g>_LF = 4.30 ; <nu_g>_HF = 4.07 ; <nu_g>_tot = 8.37
Gamma Energies:   <Eg>_LF = 0.76 MeV ; <Eg>_HF = 0.74 MeV ; <Eg>_tot = 0.75 MeV

//// THE END ////
```

1.3 Physics Models Implemented in CGMF

CGMF implements the Hauser-Fesbach theory of statistical nuclear reactions applied to the de-excitation of the fragments produced in a binary fission reaction. It uses the Monte Carlo approach to produce an event-by-event view of the process, following each sampled decay path exactly, i.e., conserving energy, spin and parity as the excitation energy is released through the emission of neutrons and photons. To perform this calculation, several physics models have been incorporated into CGMF, as well as input parameter and/or input data. Those are discussed in this Section.

1.3.1 Fission Fragment Yields

CGMF does not calculate the initial pre-neutron fission fragment yields. Instead, it reads or reconstructs those yields in mass, charge and kinetic energy, $Y(A, Z, TKE)$, from experimental data or systematics. Several theoretical efforts are underway to predict fission fragment yields from dynamical fission calculations. We will incorporate the results of those works as they become available and more accurate.

In the present version of **CGMF**, only binary fission events are considered. Ternary fission in which an α particle is emitted along with the two fragments is not treated, nor more complicated "fission" splitting, e.g., accompanied with cluster emission. In addition, the neutron emission is assumed to happen only once both fragments are *fully accelerated*. In other words, no *scission* neutrons are considered at this point. However, multi-chance fission processes such as $(n, n'f)$, $(n, 2nf)$, etc., as well as pre-equilibrium neutron emissions are taken into account at higher incident energies.

Systematics of scission fragment yields, i.e., right after scission but before any neutron emission, in mass, charge and kinetic energy, $Y(A, Z, TKE)$, have been developed for **CGMF**. They are reconstructed from partial, one-dimensional distributions.

Note: In release 1.1.0, the following fission fragment yields have been developed:

- Spontaneous fission reactions of Pu-238,240,242,244 and Cf-252,254.
- Neutron-induced fission reactions from thermal up to 20 MeV for: n+U-233,234,235,238 and n+Pu-239,241.

Mass Yields

By default, **CGMF** implements a three-Gaussian model to represent pre-neutron fission fragment mass distributions, as follows:

$$Y(A;E_n) = G_0(A) + G_1(A) + G_2(A), \tag{1.1}$$

where G_0 corresponds to a symmetric mode,

$$G_0(A) = \frac{W_0}{\sigma_0\sqrt{2\pi}}\exp\left(-\frac{(A-\overline{A})^2}{2\sigma_0^2}\right),$$

and G_1 and :math‘G_2‘ are the two asymmetric modes

$$G_{1,2}(A) = \frac{W_{1,2}}{\sigma_{1,2}\sqrt{2\pi}}\left[\exp\left(-\frac{(A-\mu_{1,2})^2}{2\sigma_{1,2}^2}\right) + \exp\left(-\frac{(A-(A_p-\mu_{1,2}))^2}{2\sigma_{1,2}^2}\right)\right].$$

Here, $\overline{A} = A_p/2$, with A_p the mass of the parent fissioning nucleus, which can differ from the original compound nucleus (Z_c, A_c) if pre-fission neutrons are emitted. The parameters μ_i are the means, W_i are the weights, and σ_i are the widths of the Gaussian modes. In the case of pre-fission neutron emission, the parameters are chosen for the resulting parent fissioning nucleus (i.e. $A_p = A_0 - \nu_{\rm pre}$).

Fragment mass yields depend on the incident neutron energy, and this is reflected in the means, widths and weights of the Gaussian modes as follows:

$$\mu_i = \mu_i^{(0)} + \mu_i^{(1)}E_n$$

$$\sigma_i = \sigma_i^{(0)} + \sigma_i^{(1)}E_n.$$

and

$$W_i = \frac{1}{1+\exp[(E_n - W_i^{(0)})/W_i^{(1)}]}.$$

The weight of the symmetric Gaussian W_0 is inferred by normalizing the integrated distribution to 2.0 as: $W_0 = 2 - 2W_1 - 2W_2$.

The fission fragment mass yields obtained for Cf-252(sf) and for U-235(n,f) are shown in Figs. (1.1) and (1.2).

Charge Yields

The charge distributions as function of fragment mass and incident neutron energy follow Gaussian forms according to Wahl's systematics (Wahl:2002). The means are given by the unchanged charge distribution (UCD) corrected for the observed charge polarization. The widths are a function of the fragment mass, fitted to available experimental data as a function of energy. Shell effects, and their gradual vanishing as a function of excitation energy, are included as a function of fragment mass, fissioning nucleus, and incident neutron energy. The full description of Y(Z;A,En) can be found in Ref. (Wahl:2002).

An example of the resulting total charge yield distribution Y(Z) is shown in Fig. (1.3) for the thermal neutron-induced fission of U-235.

Charge Yields

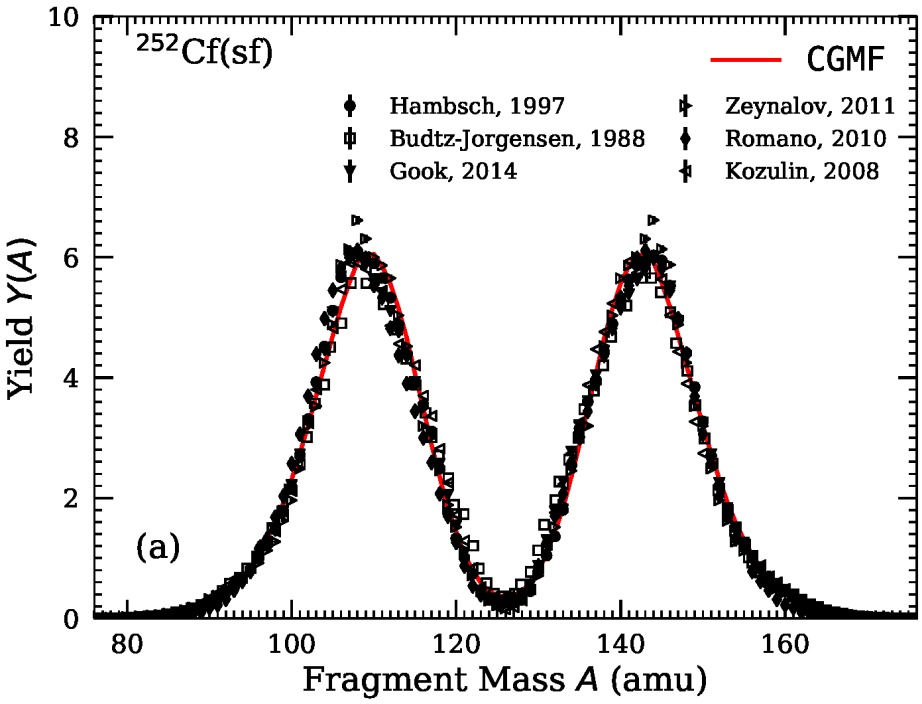


Fig. 1.1 - : Scission fragment mass yields for the spontaneous fission reaction of ^{252}Cf.

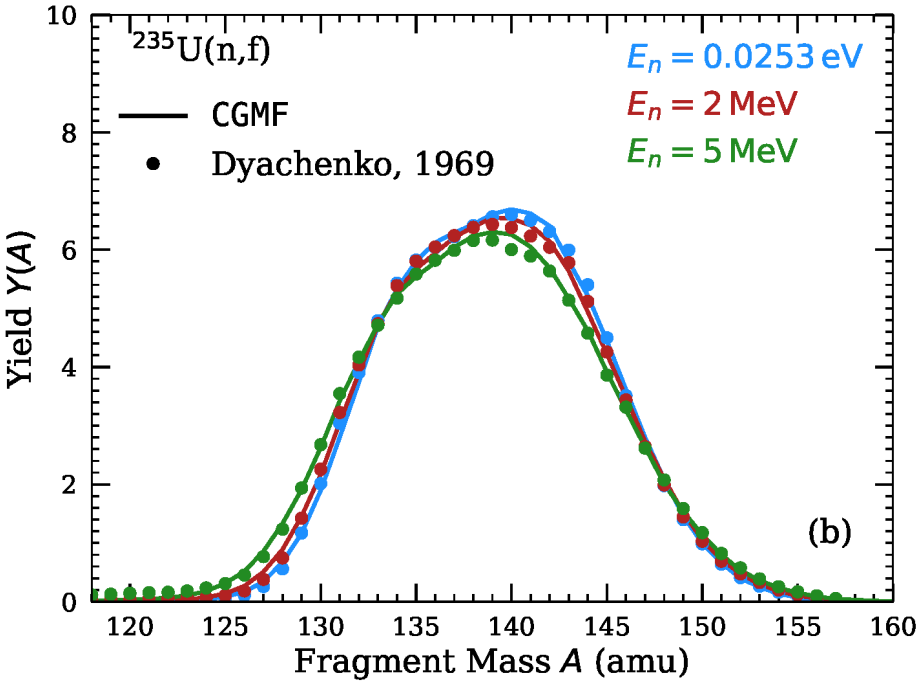


Fig. 1.2 - : Scission fragment mass yields for the neutron-induced fission reaction on ^{235}U for several incident neutron energies.

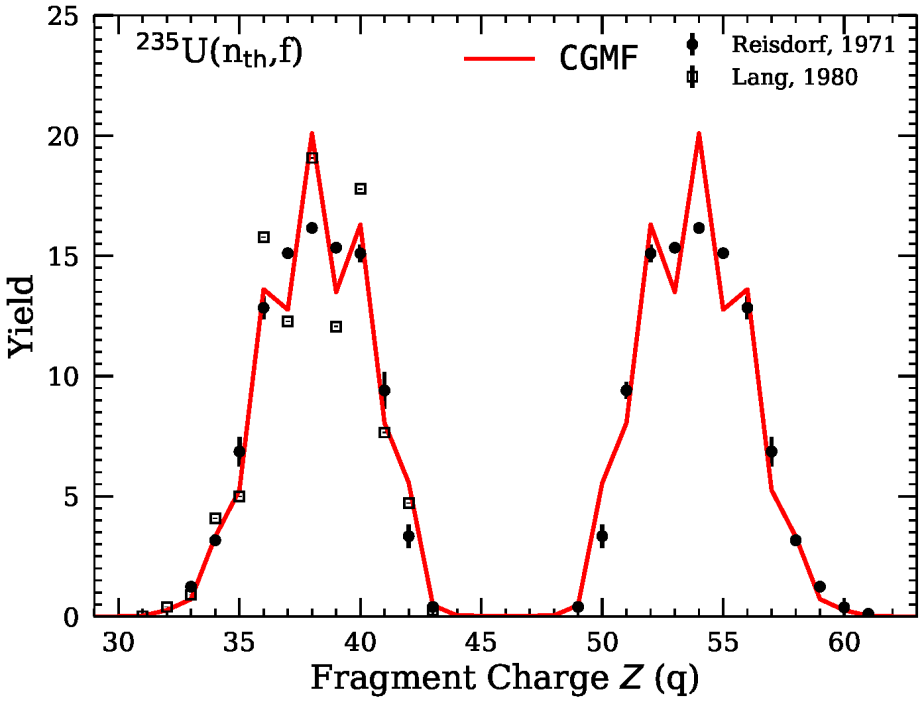


Fig. 1.3 - : Fission fragment charge yield Y(Z) for the thermal-neutron-induced fission of U-235.

Warning: Obsolete?

Wahl systematics (Wahl, 2002) are then used to obtain the charge distribution for a given mass following:

$$P(Z|A) = \frac{1}{2}F(A)N(A)\left[erf(V) - erf(W)\right], \tag{1.2}$$

where

$$V = \frac{Z - Z_p + 0.5}{\sigma_z\sqrt{(2)}} \text{ and } W = \frac{Z - Z_p - 0.5}{\sigma_z\sqrt{(2)}}$$

and $erf(x)$ represents the error function. The factor $N(A)$ is simply a normalization factor. The most probable charge is given by

$$Z_p = A_h\frac{Z_c}{A_c} + \Delta Z, \tag{1.3}$$

where Z_c, A_c are the charge and mass of the fissioning compound nucleus, σ_z is the charge width parameter and ΔZ is the charge deviation. The odd-even factor $F(A)$ is computed as

$$\begin{aligned}
F(A) &= F_Z \times F_N && \text{for } Z \text{ even and } N \text{ even}\\
F(A) &= F_Z/F_N && \text{for } Z \text{ even and } N \text{ odd}\\
F(A) &= F_N/F_Z && \text{for } Z \text{ odd and } N \text{ even}\\
F(A) &= 1/(F_Z \times F_N) && \text{for } Z \text{ odd and } N \text{ odd}
\end{aligned}$$

The average charge distribution is obtained by convoluting $Y(Z|A)$ over the fragment mass distribution $Y(A)$, and the result is shown in figure *fig-YZ-Einc* for the heavy fission fragments only.

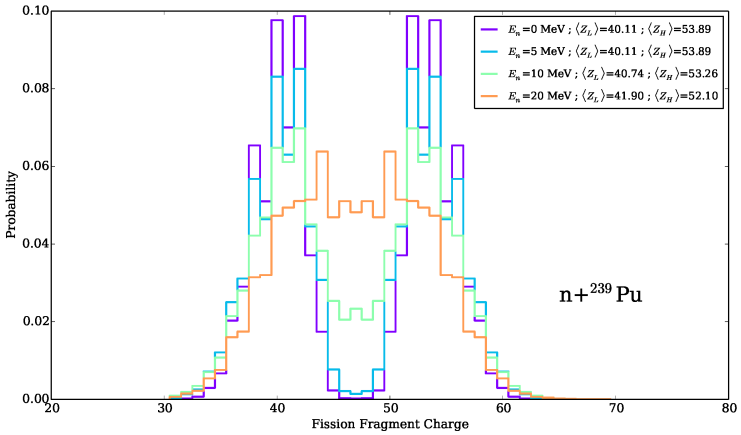


Fig. 1.4 - : Fission fragment charge distribution as a function of incident neutron energy for the Pu-239 (n,f) reaction.

Total Kinetic Energy (TKE) Distributions

The average total kinetic energy $\overline{TKE}$ is an important quantity that determines in great part the total excitation energy available in the system for the evaporation of neutrons and photons. Since most neutrons are emitted prior to photon emission, the average total prompt neutron multiplicity, $\overline{\nu}$, strongly depends on an accurate value for $\overline{TKE}$. For the simulation of single fission events, TKE distributions have to be known for all fragments.

For thermal neutron-induced fission reactions on important isotopes as well as spontaneous fission, some reliable and rather consistent experimental data exist, albeit less so in the symmetric region where fission events are rare.

To reconstruct the total kinetic energy dependence of the fission fragment yields, one can use experimental information on the average TKE as a function of the fragment mass A as well as its width $\sigma_{TKE}(A)$. Continuing on the example above for thermal neutron-induced fission of Pu-239, we have performed a least-square fit of $\overline{TKE}(A)$ as seen in Fig. *fig-TKEA*.

The TKE distribution for each fragment mass is then reconstructed using

$$P(TKE|A) = \left(2\pi\sigma^2_{TKE}(A)\right)^{-1/2} \times \exp\left[-\frac{\left[TKE - \overline{TKE}(A)\right]^2}{2\sigma^2_{TKE}(A)}\right].$$

In a first approximation, one can assume that the shape of $\overline{TKE}(A)$ as well as $\sigma_{TKE}(A)$ are independent of the particular fissioning system and the energy of the incident neutron (see Fig. *fig-TKEA-Isotopes*). We therefore assume that only the absolute scaling of $\overline{TKE}$ changes with energy.

Note: The mass-dependent average total kinetic energy does change with incident energy, reflecting changes in the shell corrections as the excitation energy is increased. A more refined treatment of this quantity will be tackled in the future.

The energy-dependence of $\overline{TKE}$ is poorly known for most systems. However, recent experimental data have shed some light on this issue. In the current version of the code, we assume that for each pair of fission fragments, TKE can be represented by a normal distribution $\mathcal{N}_{(\langle TKE \rangle, \sigma_{TKE})}(A, E_n)$, and assume that the energy dependence is entirely encoded in the average value $\overline{TKE}$.

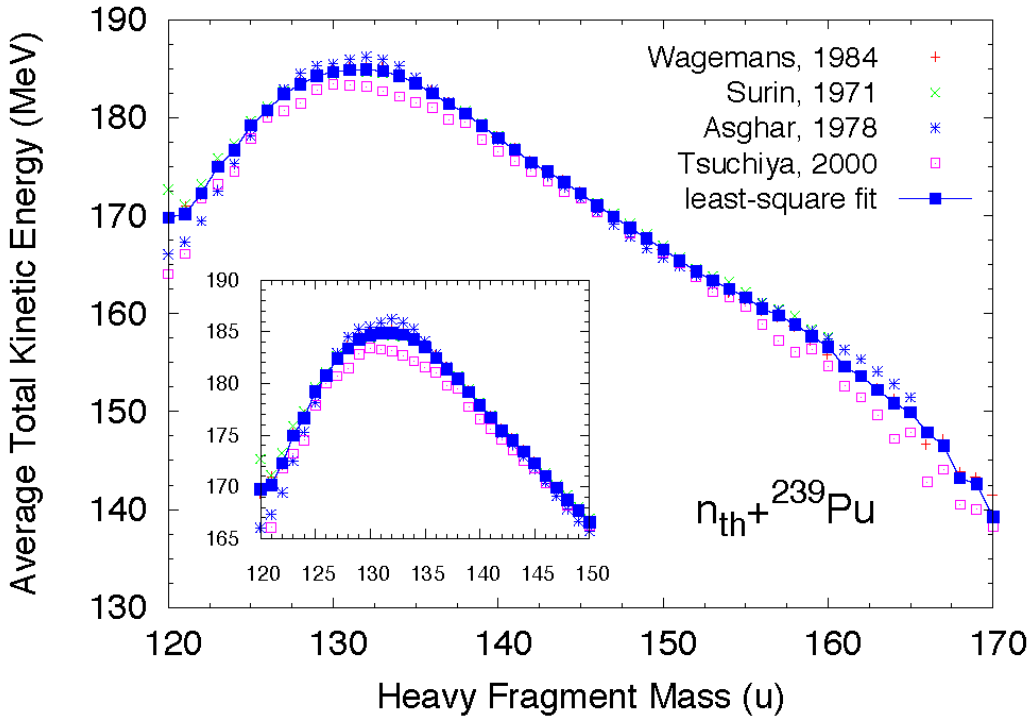
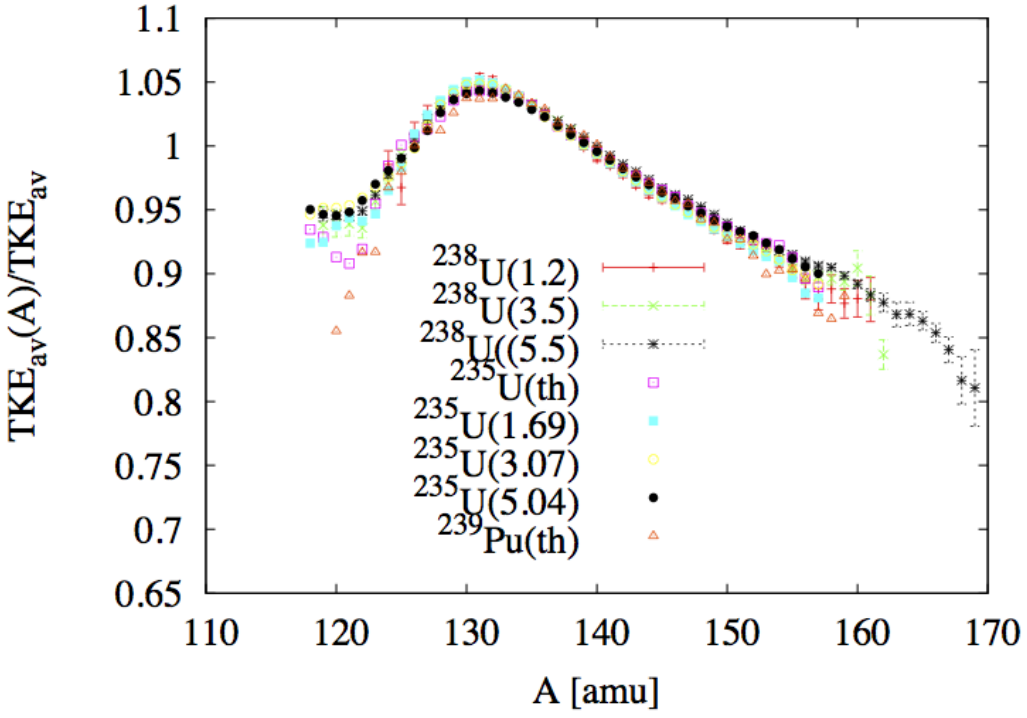


Fig. 1.5 - : Average total kinetic energy as a function of the heavy fragment mass in the case of the thermal neutron-induced fission of Pu-239.



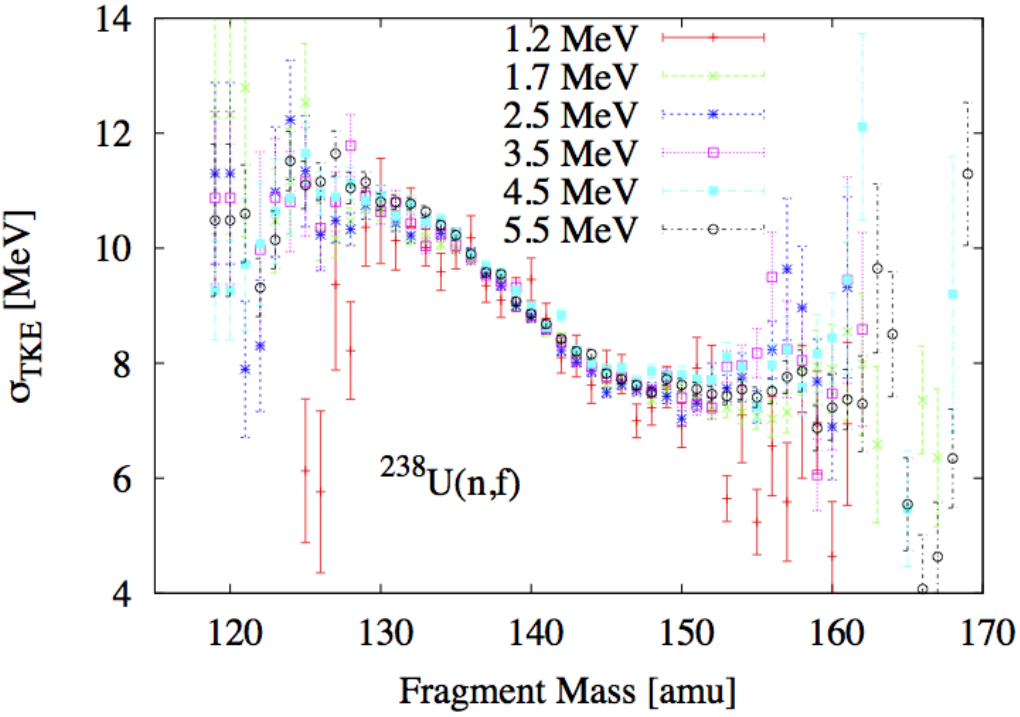


Fig. 1.6 - : Experimental data available for the mass and incident energy dependence of $\overline{TKE}$ and σ_{TKE} are shown for several fissioning systems and incident neutron energies.

In the current code implementation, the mass and energy-dependent distributions $TKE(A, E_n)$ are obtained as

$$\overline{TKE}(A, E_n) = \overline{TKE}(A, E_{th}) \times \frac{\overline{TKE}(E_n)}{\sum_A Y(A, E_n)\overline{TKE}(A, E_{th})}$$

The energy dependence of $\overline{TKE}(A)$ is given by the Madland systematics (Madland,2006), which are simple linear or quadratic fits to experimental data for selected isotopes. Making the distinction between the total fission fragment (pre-neutron) kinetic energy, TKE_{pre}, and the total fission product (post-neutron) kinetic energy, TKE_{post}, those systematics read:

For **n+U-235**,

$$\begin{aligned} TKE_{pre} &= (170.93 \pm 0.07) - (0.1544 \pm 0.02)E_n \text{ (MeV)}, \\ TKE_{post} &= (169.13 \pm 0.07) - (0.2660 \pm 0.02)E_n \text{ (MeV)}. \end{aligned} \tag{1.4}$$

For **n+U-238**,

$$\begin{aligned} TKE_{pre} &= (171.70 \pm 0.05) - (0.2396 \pm 0.01)E_n + (0.003434 \pm 0.0004)E_n^2 \text{ (MeV)}, \\ TKE_{post} &= (169.8 \pm 0.05) - (0.3230 \pm 0.01)E_n + (0.004206 \pm 0.0004)E_n^2 \text{ (MeV)}. \end{aligned} \tag{1.5}$$

And for **n+Pu-239**,

$$\begin{aligned} TKE_{pre} &= (177.80 \pm 0.03) - (0.3489 \pm 0.02)E_n \text{ (MeV)}, \\ TKE_{post} &= (175.55 \pm 0.03) - (0.4566 \pm 0.02)E_n \text{ (MeV)}. \end{aligned} \tag{1.6}$$

Madland's fits were only constructed up to the threshold for second-chance fission. We assume however that they are valid at higher energies as well for the initial fissioning nucleus. Above the second-chance fission threshold, the

average TKE does not necessarily follow a linear or quadratic behaviour though, as successive neutron emissions modify the fissioning nucleus and its excitation energy. We further assume that Madland's energy-dependence parameterizations remain valid for the nuclei A-1, A-2, etc. Only the reference thermal value of $\overline{TKE}(E_{th})$ is changed according to Viola's systematics (Viola:1985])

$$\overline{TKE}_{th} = (0.1189 \pm 0.011)\frac{Z^2}{A^{1/3}} + (7.3 \pm 1.5)\ \text{MeV}. \tag{1.7}$$

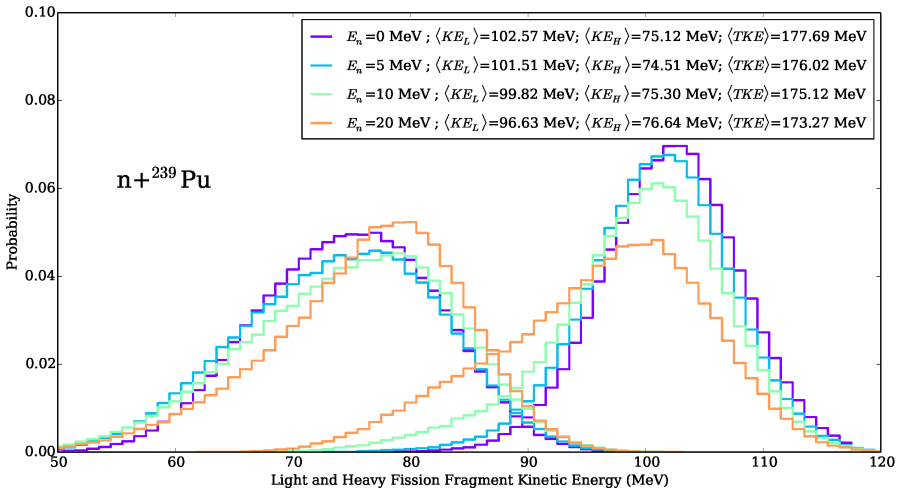


Fig. 1.7 - : Fission fragment kinetic energy distribution as a function of incident neutron energy for the Pu-239 (n,f) reaction.

Complete $Y(A, Z, TKE)$ Yields Reconstruction

Finally, the full pre-neutron emission fission fragment distributions can be reconstructed as:

$$Y(A, Z, TKE) = Y(A) \times P(Z|A) \times P(TKE|A) \tag{1.8}$$

The resulting $Y(A, TKE)$ distribution is shown here:

The approach described above to evaluate the pre-neutron emission fission fragment yields is not unique, and depends on the type of experimental data that have been measured. In some cases, the two-dimensional $Y(A, TKE)$ distribution has been measured (Romano,2010), and therefore only the charge distribution for every fragmentation has to be computed to obtain the full distribution. In the majority of cases, however, no such information is available and one has to rely on systematics and/or phenomenological models. The present version of **CGMF** is limited to the few isotopes and reactions that have been well measured. The extension to other isotopes and reactions is planned for the near future.

1.3.2 Initial Excitation Energy, Spin and Parity Distributions

The total excitation energy (TXE) available to the two fragments is constrained by the energy conservation rule

$$\begin{aligned} TXE &= Q_f - TKE, \\ &= E_{inc} + B_n + M_n(A_f, Z_f)c^2 - M_n(A_1, Z_1)c^2 - M_n(A_2, Z_2)c^2 - TKE \end{aligned} \tag{1.9}$$

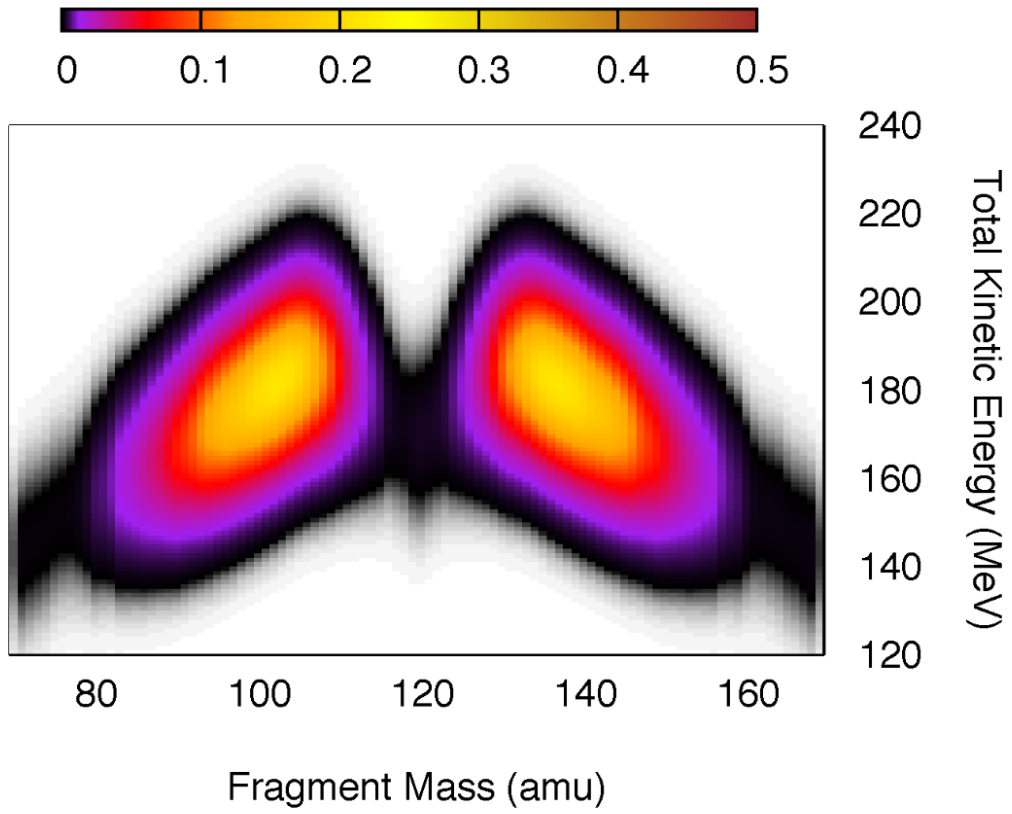


Fig. 1.8 - : Mass and Total Kinetic Energy yields reconstructed using Eq. (1.8) in the thermal neutron-induced fission of Pu-239.

where TKE is the total kinetic energy, i.e. the sum of the kinetic energies of fragment 1 and fragment 2, and M_n are the nuclear masses for the fissioning nucleus, and the fragments 1 and 2 respectively. Once TKE is known, the total excitation energy TXE is also known. However, the partitioning of this energy between the two fragments is a more complicated matter, which is discussed at more length in the section below.

Excitation Energy Partitioning

As mentioned above, the total excitation energy (TXE) is known as long as the total kinetic energy (TKE) and nuclear masses are known. What is not completely known however is the way TXE is distributed among the light and the heavy fragments.

Several interesting and competing ideas have been proposed to explain how TXE is shared among the two fragments (Schmidt,2010) (Talou,2011), but no fully compelling proof has been given so far supporting those theories. They all rely on some assumptions regarding the configurations of the fission fragments near the scission point. In the present version of **CGMF**, this excitation energy partitioning is treated as a free parameter, which can be tuned to be best reproduce the average prompt fission neutron multiplicity as a function of the fragment mass, $\overline{\nu}_p(A)$. Indeed, to the first order, the neutron multiplicity reflects the excitation energy of the fragment, while the average neutron energy reflects the temperature of the fragment.

We introduce the ratio of the temperatures between the light and heavy fragments:

$$R_T = \frac{T_l}{T_h}, \tag{1.10}$$

and use the Fermi gas formula to infer the sharing of the excitation energy. This ratio parameter depends on the fragment pair masses A_l and A_h. At this stage, it is only a convenient way to parameterize the partitioning of TXE, and nothing more. Note that this parameter can also be confusing as it uses a ratio of temperatures, while its correct purpose is to share excitation energies. It was introduced at first in the context of the Los Alamos model (LAM) (Madland,1982) to compute the average prompt fission neutron spectrum. In its original formulation, the LAM uses a distribution of temperatures to represent the intrinsic excitations in the fragments, and uses the same distribution for both the light and the heavy fragments. In other words, $R_T = 1.0$.

In **CGMF**, R_T can be chosen to be mass-dependent to best reproduce $\overline{\nu}_p(A)$. In most cases, it means that $R_T > 1.0$ as more excitation energy is pumped into the light fragment at the expense of the heavy fragment. This result is in large part due to the deformation energies of the nascent fragments, the heavy fragment being closer to a sphere thanks to shell closures, while the light fragment is largely deformed. This is not true everywhere however, and for very asymmetric fragmentations the inverse becomes true.

We are working on a more physically and mathematically sound proof of this empirical result, in particular in order to expand **CGMF** calculations to other isotopes and energies more reliably.

Figure *fig-Ui* shows an example of a distribution of initial excitation energies in the light and heavy fragments, as well as the total energy, in the case of Cf-252 spontaneous fission.

Spin and Parity Distributions

The spin of the fragments also follows a conservation rule

$$\vec{J_1} + \vec{J_2} + \vec{l} = \vec{J_f} \tag{1.11}$$

where $\vec{J_1}$ and $\vec{J_2}$ are the fission fragment total spins, $\vec{J}$ is the total angular momentum of the fissioning nucleus, and $\vec{l}$ is the relative orbital angular momentum between the two fragments. In the present version of **CGMF**, $\vec{J_1}$ and $\vec{J_2}$ follow a Gaussian distribution around a mean value that is chosen to best reproduce some of the observed prompt photon characteristics. The relative orbital angular momentum l is left free, so there is no correlation between $\vec{J_1}$ and $\vec{J_2}$ at

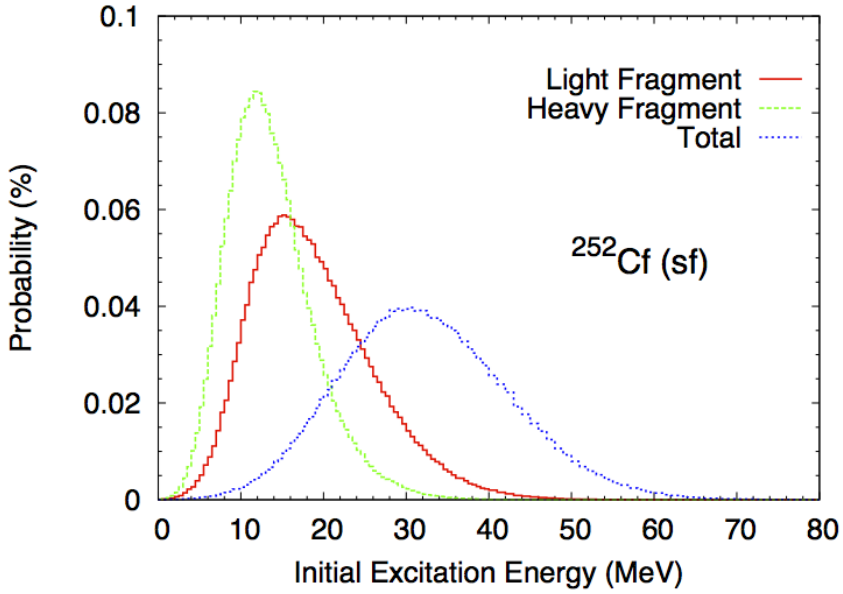


Fig. 1.9 - : Typical initial excitation energy distributions in the light and heavy fragments, as well as the total, computed in the case of Cf-252 spontaneous fission.

this point. This question will be revisited in future versions of the code. Also, negative and positive parities are chosen to be equally probable, so the spin and parity distribution in the fragments reads

$$\rho(J,\pi) = \frac{1}{2}(2J+1)\exp\left[-\frac{J(J+1)}{2B^2(Z,A,T)}\right] \tag{1.12}$$

where B is defined in terms of the fragment temperature as

$$B^2(Z,A,T) = \alpha\frac{\mathcal{I}_0(A,Z)T}{\hbar^2},$$

and $\mathcal{I}_0(A,Z)$ is the ground-state moment of inertia of the fragment (A,Z). α is an adjustable parameter that is used globally to reproduce prompt fission γ data.

Typical values calculated for the light and heavy fragments are 6-8 $\hbar$, in rather good agreement with values cited in the literature– see (Wilhelmy,1972) for instance.

1.3.3 Hauser-Feshbach Statistical Decay

The Hauser-Feshbach theory (Hauser-Feshbach,1952) describes the decay of a compound nucleus in statistical equilibrium through the evaporation of particles and photons until a ground-state or long-lived isomer is reached. This is schematically represented in Fig. 2.6.

In this schema, a fragment (A,Z) is represented by its ground-state at energy zero, a set of low-lying discrete excited states, and by a set of energy-bins at higher excitation energy where the density of levels becomes too high for individual levels to be separated experimentally. In practice, this picture is not a clear-cut between resolved and unresolved levels. Some levels may have been identified above the continuum threshold region, but it may also be known, from a statistical analysis of the observed levels, that a significant portion of levels has not been observed or that a large fraction of observed levels could not be assigned a specific spin or/and parity. In this case, the matching energy between the discrete and continuum regions is often lowered to well-known levels.

Fission fragments are neutron-rich, and often relatively far from the valley of $beta$-stability where most experiments have been performed. The known spectroscopy of neutron-rich nuclei is very poor compared to stable nuclei, which

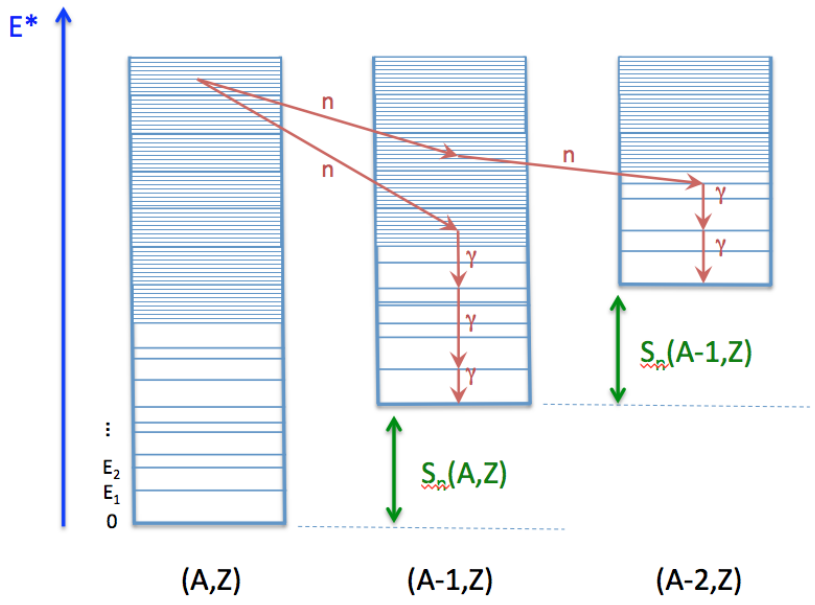


Fig. 1.10 - : Schematic drawing explaining the representation of a nucleus in the **CGMF** code, and individual decay paths followed through Monte Carlo simulations.

means that often very few discrete levels are known. In this case, the matching of the discrete region to the continuum is complicated and very sensitive to the number of specific levels included in the analysis. One also has to rely on systematics of level density parameters to describe the continuum region. Those systematics have been established for stable nuclei and large uncertainties can be expected in the description of nuclei far from stability.

In Fig. 2.6, a couple of decay paths, starting from the same initial excitation energy-bin, are drawn (red arrows) to illustrate the emission of neutrons and photons. In a traditional deterministic Hauser-Feshbach reaction code, the daughter nuclei are all populated at the same time. In a Monte Carlo code such as **CGMF**, only one path is chosen at a given step.

The Hauser-Feshbach theory is statistical in nature and the decay paths are governed by the probabilities for the system to evolve in a particular reaction channel that is open, i.e. physically possible given constraints in energy, spin and parity. We will denote a channel c by:

$$c \equiv (A_i, Z_i, U_i, J_i, \pi_i; A_f, Z_f, U_f, J_f, \pi_f)$$

In the case of neutron or photon emissions only, we always have $Z_i = Z_f$, and $A_i = A_f$ (photon) or $A_f = A_i - 1$ (neutron).

The probability of decaying through a particular channel c is given by the product of the channel transmission coefficients and the density of levels in the final state. For photons, we have:

$$P(\epsilon_\gamma)dE \propto T_\gamma(\epsilon_\gamma)\rho(Z, A, E - \epsilon_\gamma)dE,$$

and for neutrons

$$P(\epsilon_n)dE \propto T_n(\epsilon_n)\rho(Z, A-1, E - \epsilon_n - S_n)dE,$$

where ϵ_γ and ϵ_n are the center-of-mass energies of the emitted photon and neutron, respectively.

1.3.4 Neutron Emission Probabilities

Neutron transmission coefficients $T_n^{lj}(\epsilon)$ are obtained through optical model calculations. In this model, the Schroedinger equation describing the interaction of incoming waves with a complex mean-field potential is solved, providing the total, shape elastic and reaction cross-sections. It also provides the transmission coefficients that are used in the compound nucleus evaporation calculations.

The transmission coefficients for a channel c are obtained from the scattering matrix S as

$$T_c = 1 - |\langle S_{cc} \rangle|^2 .$$

To calculate the neutron transmission coefficients for fission fragments, it is important to rely on a global optical model potential (OMP) that can provide results for all nuclei. By default, **CGMF** uses the global spherical OMP of Koning and Delaroche.

It is important to note that the calculated spectrum of prompt neutrons does depend on the choice of the optical potential used to compute the neutron transmission coefficients. The OMP of Koning-Delaroche has been established to describe a host of experimental data, e.g., total cross-sections, S_0 and S_1 strength functions, etc. However, those data are only available for nuclei near the valley of stability. Some experimental information do indicate that this optical potential may not be very suitable to the fission fragment region, and therefore a relatively large source of uncertainty in the calculation of the neutron spectrum results from this open question.

Pre-Fission Neutrons

If the initial excitation energy in the compound nucleus is high enough, there is a chance that neutrons are evaporated prior to fission. We then talk about first-chance (n, f), second-chance $(n, n'f)$, third-chance $(n, 2nf)$, etc., fissions. The probabilities for each multi-chance fission event to occur can be computed from the Γ_n/Γ_f ratio as a function of the incident neutron energy. This ratio depends in turn on the fission barrier heights in the various compound nuclei $A, A-1, A-2$, etc. The **CoH3.0.4** code was used to calculate those ratios for different actinides. As an example, we show here the case of n+Pu-239, in comparison with ENDF/B-VII.1 and JENDL-4.0 evaluations. The **CoH** calculations tend to predict a much higher second-chance fission probability at the expense of the first-chance, compared to the evaluations. These quantities are not observables though, and it is therefore difficult to judge about the validity of those curves at this point.

In **CGMF**, those multi-chance fission probabilities are sampled to determine the number of pre-fission neutrons. Then, the energies of those neutrons are obtained by sampling the corresponding neutron spectra. In the case of the first emitted neutron, the spectrum corresponds to a weighted sum of a pre-equilibrium and an evaporation components. The fraction of pre-equilibrium neutrons is also calculated in the **CoH** code using the exciton model. Then, the first neutron-out spectrum is given by:

$$\chi_1 = f_{pe}\chi_{pe} + (1 - f_{pe})\chi_{evap}.$$

The energy-dependent fraction f_{pe} can be fitted by a simple function:

$$f_{pe}(E_{inc}) = \frac{1}{1 + \exp\left[(12.49 - E_{inc})/10.21\right]} - 0.042E_{inc} - 0.25.$$

As can be seen in Fig. *fig-PE*, it is a very reasonable approximation for neutron-induced reactions on U-235, U-238 and Pu-239.

1.3.5 Gamma-Ray Emission Probabilities

The γ-ray transmission coefficients are obtained using the strength function formalism from the expression:

$$T^{Xl}(\epsilon_\gamma) = 2\pi f_{Xl}(\epsilon_\gamma)\epsilon_\gamma^{2l+1},$$

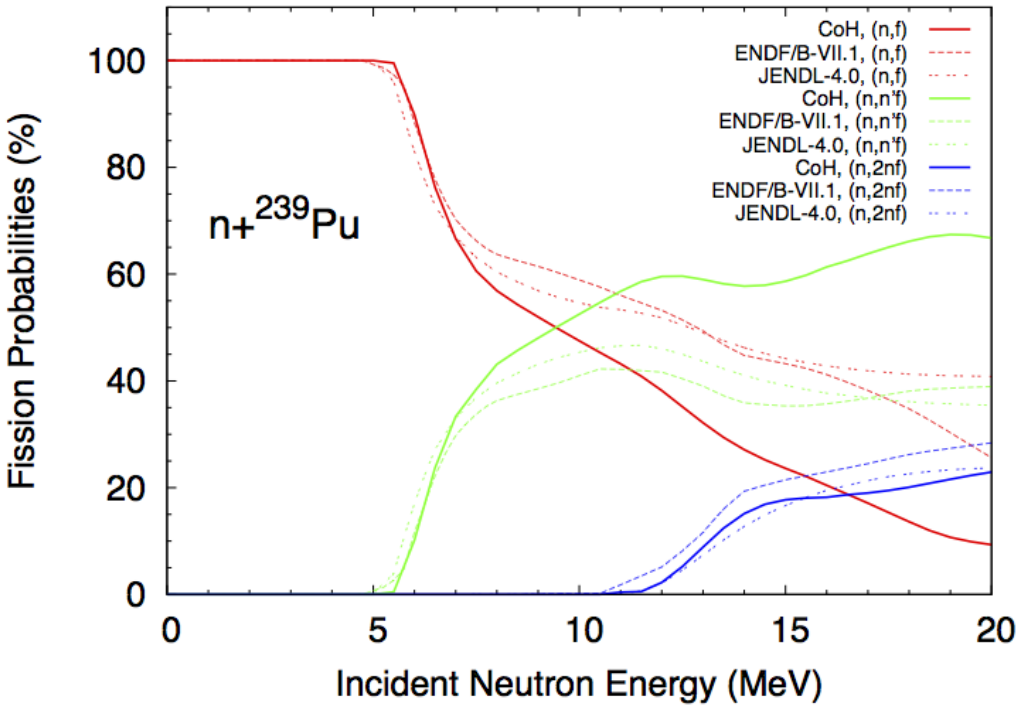


Fig. 1.11 - : Multi-chance fission probabilities in the neutron-induced fission reaction on Pu-239 as calculated with the **CoH** code (and used in **CGMF**), and in comparison with the ENDF/B-VII.1 and JENDL-4.0 evaluations.

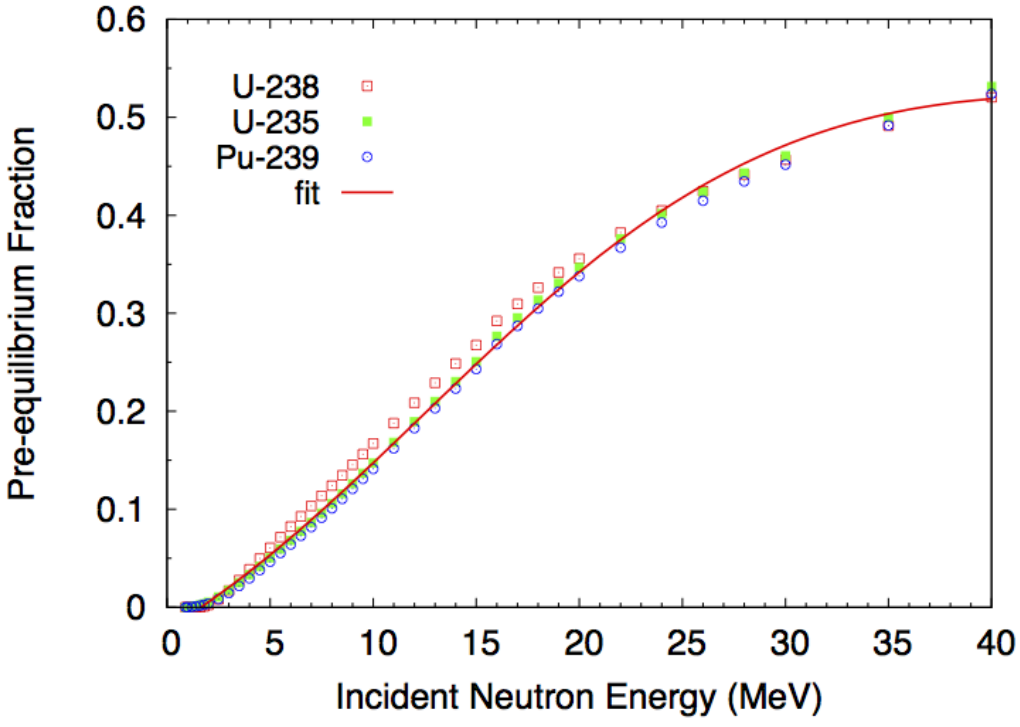


Fig. 1.12 - : Pre-equilibrium fractions calculated with the **CoH** code. There is only a slight dependence on the target nucleus, and the fit formula (solid line) is used by default in **CGMF** instead.

where ϵ_γ is the energy of the emitted gamma ray, Xl is the multipolarity of the gamma ray, and $f_{Xl}(\epsilon_\gamma)$ is the energy-dependent gamma-ray strength function.

For $E1$ transitions, the Kopecky-Uhl generalized Lorentzian form for the strength function is used:

$$f_{E1}(\epsilon_\gamma, T) = K_{E1}\left[\frac{\epsilon_\gamma \Gamma_{E1}(\epsilon_\gamma)}{\left(\epsilon_\gamma^2 - E_{E1}^2\right)^2 + \epsilon_\gamma^2 \Gamma_{E1}(\epsilon_\gamma)^2} + \frac{0.7\Gamma_{E1} 4\pi^2 T^2}{E_{E1}^5}\right]\sigma_{E1}\Gamma_{E1}$$

where σ_{E1}, Γ_{E1}, and E_{E1} are the standard giant dipole resonance (GDR) parameters. $\Gamma_{E1}(\epsilon_\gamma)$ is an energy-dependent damping width given by

$$\Gamma_{E1}(\epsilon_\gamma) = \Gamma\frac{\epsilon_\gamma^2 + 4\pi^2 T^2}{E_{E1}^2},$$

and T is the nuclear temperature given by

$$T = \sqrt{\frac{E^* - \epsilon_\gamma}{a(S_n)}}.$$

The quantity S_n is the neutron separation energy, E^* is the excitation energy of the nucleus, and a is the level density parameter. The quantity K_{E1} is obtained from normalization to experimental data on $2\pi\langle\Gamma_{\gamma 0}\rangle/\langle D_0\rangle$.

For $E2$ and $M1$ transitions, the Brink-Axel (Brink,1955) (Axel,1962) standard Lorentzian is used instead:

$$f_{Xl}(\epsilon_\gamma) = K_{Xl}\frac{\sigma_{Xl}\epsilon_\gamma\Gamma_{Xl}^2}{(\epsilon_\gamma^2 - E_{Xl}^2)^2 + \epsilon_\gamma^2\Gamma_{Xl}^2}.$$

In the current version of **CGMF** (ver. 1.1), only $E1, E2$, and $M1$ transitions are allowed, and higher multipolarity transitions are neglected.

1.3.6 Continuum Level Densities

In **CGMF**, the Gilbert-Cameron model of level densities is used for all fragments. In this model, a constant temperature formula is used to represent the level density at lower excitation energies, while a Fermi gas formula is used at higher excitation energies. Experimental data on the average level spacing at the neutron separation energy can be used to constrain parameters entering the Fermi gas formula, while low-lying discrete levels are used to constrain the constant-temperature parameters. Again, little data is available for nuclei far from stability where systematics have been developed, contribution to uncertainties in the final predicted data.

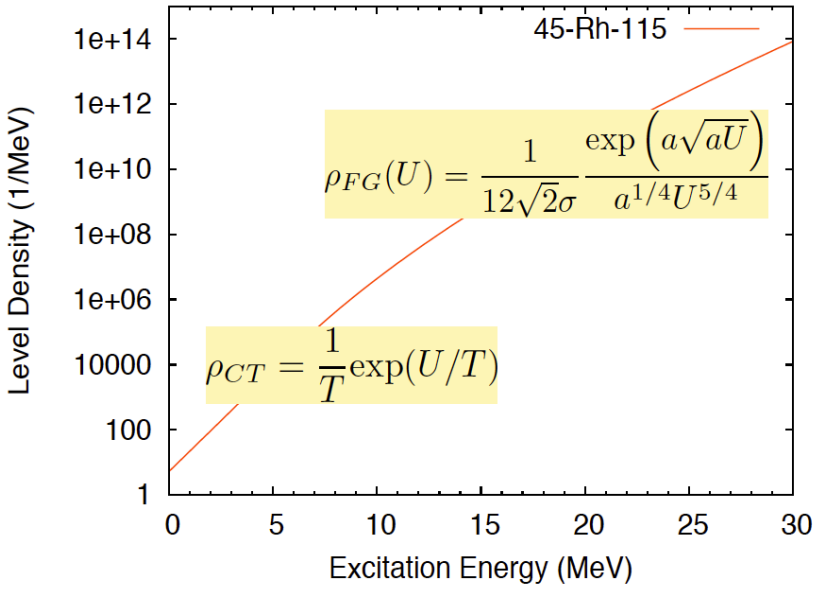
The constant temperature form is given by

$$\rho_{CT}(U) = \frac{1}{T}\exp\left(\frac{U + \Delta - E_0}{T}\right),$$

where T is the nuclear temperature and E_0 is a normalization factor. The quantity U is the excitation energy E minus the pairing energy Δ. At higher excitation energies, the Fermi gas form of the level density is used instead and is given by

$$\rho_{FG}(U) = \frac{\exp\left(2\sqrt{aU}\right)}{12\sqrt{2}\sigma(U)U(aU)^{1/4}},$$

where a is the level density parameter. The constant temperature form of the level density is matched to cumulative low-lying discrete levels, when they are known. For fission fragments, which are neutron-rich and rather poorly known, this constant-temperature level density is sometimes used down to the ground-state, as shown in the following figure



In its original formulation, the Gilbert-Cameron formalism uses an energy-independent level density parameter a. To better describe the washing-out of shell effects at higher excitation energies, Ignatyuk (Ignatyuk,1979) developed a model that uses an energy functional for the level density parameter as

$$a(U) = \tilde{a}\left(1 + \delta W \frac{1 - \exp(-\gamma U)}{U}\right).$$

In this formula, $\tilde{a}$ is the asymptotic value of the level density parameter at high energy, δW is the shell correction energy, and γ is an empirical damping width to account for the washing-out of shell effects at high energy.

1.3.7 Discrete Levels

The nuclear structure information needed to describe each fission fragment is obtained from the RIPL-3 library. Nuclear structure data are always evolving, depending on the availability of new nuclear structure experiments. **CGMF** does not calculate the excited states in a given nucleus, but instead fully depends on the ENSDF or somewhat equivalenty, RIPL3 libraries. The study of isomeric ratios or more generally specific γ decay chains strongly depends on the quality of the underlying nuclear structure information.

Preparing the nuclear structure file for CGMF

A special discrete level file (`cgmfDiscreteLevels.dat`) for use with **CGMF** is produced from the RIPL-3 library, and includes adjustments for incomplete information. The Jupyter notebook `transformLevelDataFile.ipynb` and the python class `Nucleus.py` are used for this purpose.

1. For each nucleus, the level data are first read from the RIPL-3 complete datafile;
2. The levels are then "fixed" for missing or unassigned spin and parity;
3. Finally, the gamma transitions are "fixed".

Fixing the level scheme

If the spin or/and parity of a level is negative, then the level is kept and its (J, π) values are chosen according to the distribution assumed in the continuum following the Kawano-Chiba-Koura (KCK) level density systematics [J. Nucl. Sci. and Tech. 43, 1 (2006)]. Assuming that the parities are evenly distributed, the missing level parity is chosen randomly.

In some cases, RIPL-3 would provide choices of spin and parity for a uncertain level. In that case, the first option will be selected.

In the case of the ground-state being completely unknown, default values will be selected according to:

- even-even: 0^+
- odd-odd: 1^+
- even-odd: $1/2^+$

Warning: Such arbitrary decisions can have an non-negligible impact when studying specific γ decay chains in a particular nucleus. In that case, extra caution should be put in interpreting the results of **CGMF** by studying the origin of the nuclear structure information available (or not) for this nucleus.

Fixing the γ transitions

In addition to incomplete level schemes, the nuclear structure data from RIPL-3/ENSDF can have incomplete decay chains.

In the case of the first excited state, if no decay data is available, it is assumed that the level decays 100% into the ground-state emitting a γ ray of energy corresponding to the excitation energy of this first level.

Todo: is this the right thing to do? Not always!

In the case of higher excited levels, the spin and parity are chosen according to the (J, π) values of levels below in energy, and to which the current uncomplete level could be reached through an E1 transition.

Note: More sophisticated level decay schemes should be employed.

References

1. T.Kawano, S.Chiba and H.Koura, *Phenomenological Nuclear Level Densities using the KTUY05 Nuclear Mass Formula for Applications Off-Stability*, J. Nucl. Sci. and Tech. 43, 1 (2006)

1.3.8 Isomeric States

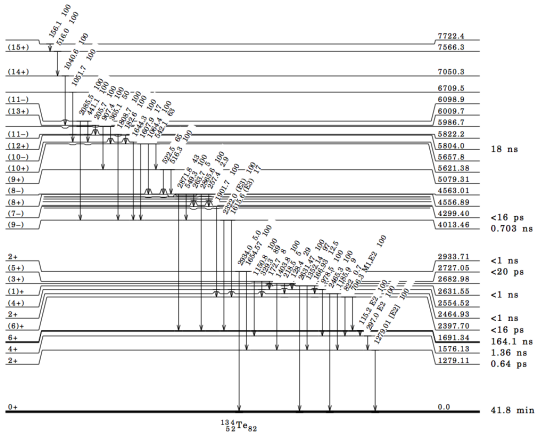
Many low-lying discrete levels that are reported in the ENSDF database have a measurable half-life, ranging from nanoseconds to seconds and even longer. **CGMF** takes this into account when calculating the gamma cascades in the fission products, and samples the exponential decay law according to the reported half-lives.

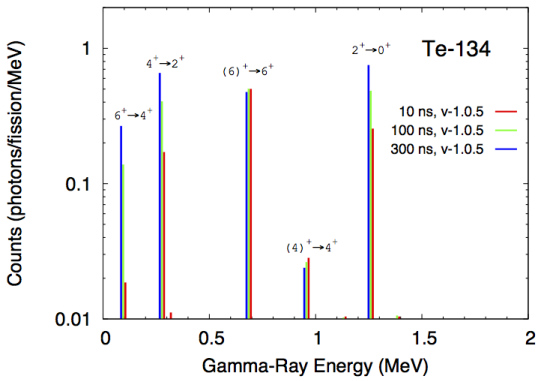
An experimental time coincidence window can be set in the `config.h` configuration file:

```
const double EXPERIMENTAL_TIME_WINDOW = 1e-8;
```

The time is given in seconds, so in the example above, 1e-8 corresponds to 10 ns. The default value is negative. In this case, all levels are set to decay to the ground-state, ignoring half-lives entirely. Since this value is stored in a configuration file, it is set at compilation time. If the user decides to change this value, he/she would need to recompile the code before using it.

As an example, the calculated intensities for specific gamma lines in Te-134, in the thermal neutron-induced fission of U-235, are shown in the figure below. Time-coincidence windows of 10, 100 and 300 ns were used in three separate calculations. Because of the presence of ~100 ns isomers in Te-134, some of these lines are more or less prominent depending on their half-lives. For example, the 6^+ state at 1.691 MeV has a half-life of 164 ns, decaying to the 4^+ state at 1.576 MeV. A too-short time gate (e.g., 10ns) cannot record this particular gamma line at 115 keV. Similarly, the decay of the 4^+ to 2^+ (297 keV) is also hindered since it depends on the decay of the higher excited 6^+ state.





1.4 Code Details

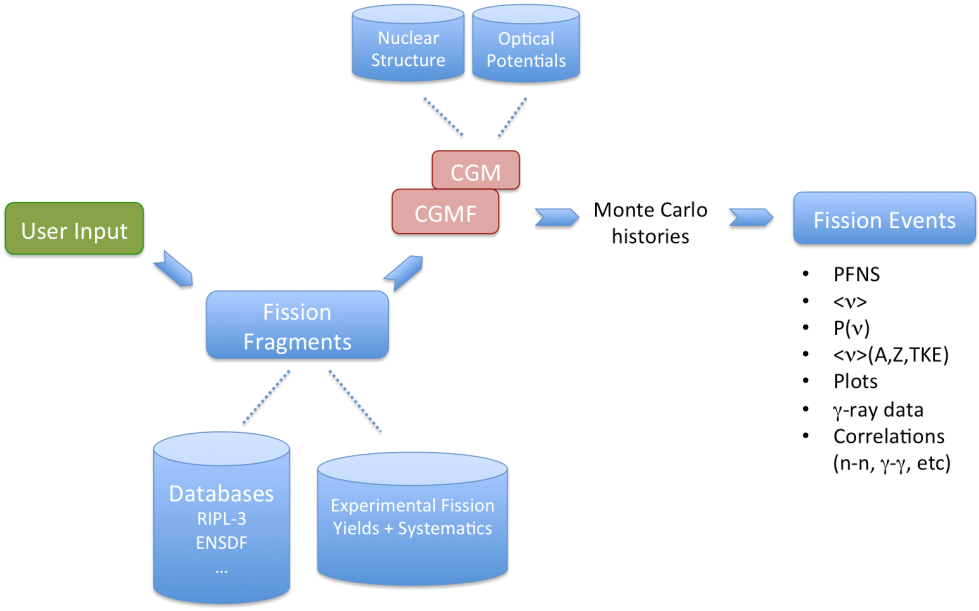
The **CGMF** code is written in C++. It is built around the **CGM** Monte Carlo statistical Hauser-Feshbach code, which provides the main computational methods for following the decay of the excited fission fragments. Two additional classes have been written out of the **FFD** code to prepare and sample the initial fission fragments, and to read out and analyze the Monte Carlo histories that are generated.

1.4.1 Code Organization & options

`cgmf.cpp` is the main driver of the code. It first reads in the user input parameters, and calls other routines from different classes to perform the requested calculations.

Algorithm

Here we describe the basic algorithm of `cgmf.cpp`.



First, the user input provided at the command line is parsed, and analyzed. The `ZAIDt` and `Einc` input parameters are mandatory. Note that some options available in **CGM** are now set fixed and hardwired for their use in **CGMF**.

Reading a history file

If the `-r` option is given, then the code reads a history file containing an ensemble of Monte Carlo histories previously generated by **CGMF**. In the case of large files, it is possible to add the `-n` option to provide a smaller number of events to be read and analyzed. In this case, the basic algorithm of the code is as follows:

```
fissionEvents = new FissionEvents (nevents);
fissionEvents->setZAIDcn (ZAIDt, Einc);
fissionEvents->readHistories (historyFile, nevents);
fissionEvents->analyzeResults ();
fissionEvents->computeFinalResults();
```

Note that all the methods used above only make use of the class `fissionEvents.cpp/.h`. A number of events (`nevents`) are read from the Monte Carlo history file using `readHistories()`, and they are analyzed in `analyzeResults()`. In this method, the characteristics of the fission fragments, the prompt neutrons and the prompt photons are stored in various histograms, e.g., `particles->Pnu[]`, `particles->nuTKE[]`, etc, where `particles` is a pointer to an `emittedParticleType`, which can be either neutrons or photons.

Finally, the method `computeFinalResults()` is used to transform histograms into average quantities, distributions and correlations among different physical quantities. For instance, the fission fragment yields are calculated and stored under `YA[]`, `YApost[]`, `YZ[]`, `YTKE[]`, `YUl[]`, etc. Prompt fission neutron and photon spectra are

calculated from the original histograms and transformed onto the outgoing energy grid `SPECTRUM_ENERGY_GRID` defined in the `init()` method in `FissionEvents.cpp`.

Performing Monte Carlo Simulations

In the more general case where one wants to perform Monte Carlo simulations of the decay of the fission fragments, the algorithm is as follows:

First, a new instance of `FissionFragments` is created to initialize the fission reaction parameters and files:

```
ff = new FissionFragments ();
```

Next, instances of light and heavy fragments are created with the total number of Monte Carlo events `nevents` specified by the user:

```
lightFragments = new fissionFragmentType [nevents];
heavyFragments = new fissionFragmentType [nevents];
```

These objects are used to track all information pertinent to each of the fragments for each fission event.

Next, a set of `nevents` fission events are produced:

```
ff->generateInitialFissionFragmentHistories (lightFragments, heavyFragments, nevents);
```

This call produces all the initial characteristics (A, Z, KE, U, J, π) for each fission fragment in each fission event.

What follows is the main loop of the program, going over every fission event and performing the de-excitation of each fission fragment:

```
// -- BEGIN LOOP OVER FISSION EVENTS ----------------------------------------
for (int ievent=0; ievent<nevents; ievent++) {
  lf = lightFragments[ievent];
  hf = heavyFragments[ievent];
  fissionEvents->addFragments (lf, hf);
  specMCMain (lf.spin, lf.parity, 0.0, 0.0, 1, spc); // light fragment calc.
  specMCMain (hf.spin, hf.parity, 0.0, 0.0, 1, spc); // heavy fragment calc.
}
// -- END LOOP OVER FISSION EVENTS ------------------------------------------
```

`lf` and `hf` are pointers to the light and heavy fission fragment partners for the fission event `ievent`. The `addFragments(lf,hf)` method is used to record the characteristics of this particular fission event. Next is the main **CGM** computational method `specMCMain()`, which performs the Monte Carlo Hauser-Feshbach calculations of the decay of this particular excited nucleus. `specMCMain()` is called twice, once for each fragment. A description of this method is given below.

Past this main loop, the Monte Carlo histories are recorded in an output file:

```
fissionEvents->writeHistories ("histories.CGMF");
```

and the results analyzed with:

```
fissionEvents->analyzeResults ();
fissionEvents->computeFinalResults();
```

This last section of the code is identical to the one used after reading a Monte Carlo history file, as explained above.

User Options

The user options, given at the command line, are as follows:

- `-i ZAIDt` : ZAID (1000*Z+A) of the target nucleus, or fissioning nucleus in the case of spontaneous fission. [required]
- `-e Einc` : energy of the incident neutron (in MeV). For spontaneous fission, set to 0.0. [required]
- `-n nevents` : number of Monte Carlo events [default: 1,000,000]
- `-r historyFile` : to read and analyze a Monte Carlo history file already produced by **CGMF**
- `-h`: display the help page for **CGMF**

Note: other options are available, but won't be described in this release of the code and user manual.

If `nevents` is negative, then only the pre-neutron emission fission fragment yields $Y(A, Z, KE, U, J, \pi)$ are produced.

If the `-r` option is given, a **CGMF** output file is read and analyzed. This is especially useful when a large output file has been generated and needs to be re-analyzed differently. In this case, the number of events can also be read, and smaller samples of the entire file can be used instead.

Configuration File(s)

CGMF comes with two configuration files, one inherited from the **CGM** code and one required specifically for fission calculations. In this manual, we describe the settings that are relevant to the fission fragment decay calculations only.

config.h

The `config.h` file is inherited from **CGM**, but with some added options. Important variables are as follows:

```
#define  DATADIR   "/usr/local/share/cgmf"
```

defines the path to the data libraries used for the Hauser-Feshbach calculations and for the initial fission fragment yields. This directory contains the RIPL-3 library of discrete levels available for many nuclei, and level density parameter systematics that are needed for fragments with unknown nuclear structure. This path should be changed by the user to reflect his/her own local data structure.:

```
const double ENERGY_BIN = 0.05;  // 50 keV
```

This constant defines the width of the energy-bin (in MeV) used in the continuum representation of the nuclear levels. A smaller value would provide a finer energy grid for the gamma-ray transitions in the continuum. For instance, if this quantity is set to 50 keV, then the minimum energy for the gamma transitions in the continuum would be 50 keV as well. Reducing this value will provide a continuum photon spectrum for energies below 50 keV, but could dramatically increase the computation time. A larger value would significantly speed up the calculations, but would cut the lower-energy part of the photon spectrum.:

```
const double CONTINUUM_LOWER_CUT = 0.02;
```

A problem inherent to all Hauser-Feshbach-type codes is the matching between the continuum and the discrete level regions describing the structure of a nucleus. The continuum region is defined by an ensemble of energy bins and a level density distribution $\rho(U_{bin}, J, \pi)$ for each energy bin. On the other hand, at lower energies, the nucleus is assumed to be fully characterized by a set of discrete levels with specific energies, spins and parities. In the course of following the decay of an excited nucleus, one sometimes populates a certain continuum energy-bin at low excitation energy, but with a high spin. The decay to this continuum state to a lower discrete level with much lower spin is strongly hampered, and can lead to an artificial series of low-energy $E1$ transitions in the continuum until a more probable low-spin transfer transition becomes available. This result is not physical. To reduce the impact of this effect, we have introduced the quantity `CONTINUUM_LOWER_CUT` to eliminate any transition below this energy.

We do not encourage users to modify this quantity, unless they know exactly what they are doing. We are working on a better solution to this problem.

```
const bool INCLUDE_INTERNAL_CONVERSION = true;
```

This boolean is set to TRUE if one wants to include the internal conversion transitions into the decay of the fragments.

```
const bool RANDOM_SEED_BY_TIME = true;
```

This boolean can be set to FALSE if one wants to fix the initial seed of the pseudo-random number generator used for the Monte Carlo samplings. This is useful in testing the reproducibility of the results, but should be set to TRUE in actual calculations.

```
const double EXPERIMENTAL_TIME_WINDOW = 1.0e-8; // 10 ns
```

This value should correspond to the experimental time coincidence window used to define the prompt fission data recorded in coincidence with a fission event. In the example above, this value is set to 10 ns. The probability of continuing a gamma cascade from an isomeric state will then depend on the value of the time window and the half-life of this isomeric state. By default, this constant is set to a negative value so that all cascades are followed until they reach the ground-state of a fission product.

config-ff.h

The `config-ff.h` is an additional configuration file, specific to fission fragment decay calculations.

```
#define MPIRUN
// #undef MPIRUN
```

CGMF can be run using MPI parallel instructions on a multi-processor machine. This can be done by commenting out the `#define MPIRUN` directive and recompiling the code. Using `#undef MPIRUN` instead would generate a non-MPI executable that is suitable to a one-processor machine.

What follows is a set of constants that define the sizes of arrays used throughout the code:

```
const int    NUMA   =  300; // number of masses A
const int    NUMZ   =  100; // number of charges Z
const int    NUMTKE =  300; // number of Total Kinetic Energy values

const int      NUME =  401; // number of energies in level density tables;
                            // dE=0.25 MeV; up to Emax=100 MeV
const double deltaE = 0.25; // energy-bin size used in level density tables

const int    NUMdZ  =   21; // [-dZ:+dZ] if dZ=10 for charge distribution
                            // around most probable Zp[A]

const int   NUMMULT =   50; // number of multiplicities

const int NUMANGLES =   73; // number of angles in angular distribution
const double dTheta =  2.5; // angular bins (degrees)

const int MAX_NUMBER_PARTICLES = 50; // max. number of particles (n or g)
                                     // emitted per fragment in a fission event

const int NUMBER_SPECTRUM_ENERGY_GRID = 641; //551;
```

Note that **none of those settings should be changed**, except by an informed user.

1.4.2 Important Classes & Methods

Note: This section needs to be updated to include the incident neutron energy dependence up to 20 MeV.

Class `FissionFragments.cpp`

This class provides all the methods and variables needed to produce the initial fission fragment yields, prior to neutron emission, characterized by a mass A, a charge Z, a kinetic energy KE, an excitation energy U, a spin J, and a parity π. The constructor is declared as:

```
FissionFragments::FissionFragments (int ZAID, float Einc, double alphaSpin);
```

In input, the user has to provide:

- the `ZAID` of the fissioning nucleus, i.e., $1000 \times Z + A$ that uniquely identifies a nucleus,
- the energy of the incident neutron, `Einc`, which should be given in MeV. In the case of spontaneous fission, 0.0 should be given.
- `alphaSpin`, which is a multiplying factor entering in the initial spin distribution of the fission fragments. Default is 1.0, which means that the original level density spin distribution for the fragments is used.

The constructor then calls the methods `setOptions()` and `init()`. The first method sets options that completely characterize the fission fragment yields $Y(A, Z, TKE)$ from partial experimental data and some systematics. It also defines the type of energy sorting mechanism allowed by the code.

Note: Below the threshold for the 2nd-chance fission, only one set of fission fragment yields have to be constructed at a particular excitation energy. At higher energies, the situation is much more complicated, as the pre-fission neutron spectrum, which includes evaporation and pre-equilibrium components, has to be sampled, leading to a residual nucleus $(A_c - \nu, Z_c, E_{res})$ that can then fission. The yields are therefore constructed "on-the-fly" while generating fission events.

The `init()` method then creates the fission fragments as:

```
fragments = new Nucleus[2];
```

where `Nucleus` is a class that fully describes a nucleus. In particular, it constructs the nuclear structure defined by a set of known low-lying discrete levels, read from the RIPL-3 database, and produces a continuum of energy bins above a certain matching energy. It also reads in nuclear masses, ground-state deformations, and individual decay transitions that are present in the database.

Class `FissionEvents.cpp`

The class `FissionEvents.cpp` provides objects and methods to read and analyze the Monte Carlo histories produced by **CGMF**. The constructor:

```
FissionEvents::FissionEvents (int maxNumberEvents) { init(maxNumberEvents); }
```

simply calls an initialization method with the maximum number of events to read and analyze. The `init()` method initializes several objects and variables: it first instantiates the objects:

```
lightFragments = new fragmentEventType [maxNumberEvents];
heavyFragments = new fragmentEventType [maxNumberEvents];
```

with the size of `maxNumberEvents`.

A `fragmentEventType` is a structure that fully characterizes a fission fragment event:

```
struct fragmentEventType {
  int    A, Z;
  double KE;  // kinetic energy (MeV)
  double Ui;  // initial excitation energy (MeV)
  int    Pi;  // initial parity
  double Ji;  // initial spin
  emissionType emissions[3]; // neutrons [0], gammas [1] and internal conversion [2]
};
```

where the `emissionType` objects are themselves defined as:

```
struct emissionType {
  int multiplicity;
  double cmEnergies   [MAX_NUMBER_PARTICLES];
  double labEnergies  [MAX_NUMBER_PARTICLES];
  double cmAngles     [MAX_NUMBER_PARTICLES];
  double labAngles    [MAX_NUMBER_PARTICLES];
  int transitionTypes [MAX_NUMBER_PARTICLES];
};
```

and fully defines a particular emission in energy, angle in both the center-of-mass and laboratory frames, and type of emission, e.g., neutron, gamma or internal conversion.

The `lightFragments` and `heavyFragments` objects are then used to store all fission event data for the total number of events (`maxNumberEvents`).

The initialization subroutine also defines the outgoing energy grid used to report the particle energy spectra. It finally initializes several storage arrays.

The following method:

```
void FissionEvents::addFragments (fissionFragmentType lf, fissionFragmentType hf) {...
↪};
```

is used to save all the data pertaining to the fission fragments (A, Z, KE, U, J, π) in a fission event, for both complementary fragments.

The decay of the fission fragments is handled by the routine `specMCMain()` (described below). Once all Monte Carlo samplings have been performed, the results are then saved into a history file by:

```
void FissionEvents::writeHistories (string outputFilename) {...};
```

This routine simply takes an filename in input, opens the file for writing, and writes all the Monte Carlo histories onto it. The resulting file can later be read and analyzed using the method:

```
void FissionEvents::readHistories (string inputFilename, int numberEvents) {...};
```

The next step is to analyze the results. This is done through:

```
void FissionEvents::analyzeResults (void) {...};
```

This routine loops over all fission events, and fills out many variables and arrays, such as `particles->lfEcm`, which records the center-of-mass energies of the emitted neutrons and photons coming from the light fragment, `particles->nuTKE[iTKE]`, which records the average particle multiplicity as a function of the total kinetic energy, `gammaMultiplicityNu[]`, which records the gamma-ray multiplicity versus neutron multiplicity correlations, etc.

Note that if **CGMF** is run with MPI parallel instructions, then `MPI_Reduce()` calls are made here.

Another routine:

```
void FissionEvents::computeFinalResults (emittedParticleType * particles) {...};
```

is used to finalize the results by transforming histograms into spectra, renormalizing yields, calculate different average quantities as a function of fragment properties, etc. Those two last routines may be merged in a future (cleaner) version of the code.

Finally, results can be saved in a format that is custom-readable by GNUPLOT scripts through:

```
void FissionEvents::saveResultsToGnuplot () {...};
```

Method `specMCMain()`

This method is at the core of **CGM/F** calculations. It performs Monte Carlo samplings of emission probabilities for all open channels following the Hauser-Feshbach [Hauser-Feshbach:1952] statistical formalism of nuclear reactions. For every initial configuration of a compound nucleus (A, Z) in excitation energy U, spin J and parity π, it prepares the nucleus by reading its known low-lying structure from the RIPL-3 database, prepares its continuum energy-bins, and compute the neutron and photon transmission coefficients. It does this for a certain number of nuclei that can be produced in multiple neutron emissions.

The method then samples the emission probability distributions, and choses one particular decay path. It records the Monte Carlo histories through `recordEmittedParticles()` for further reading and analysis by **CGMF**.

1.5 CGMFtk - python analysis package

python provides a powerful framework to analyze the output of the **CGMF** history files. We have created a python package, *CGMFtk* that can be used to read the yield file or history file created by **CGMF** and then easily extract results of interest, including calculating neutron and gamma multiplicities, prompt particle energy spectra, isomeric ratios, and correlations between observables of interest. Details on how to calculate many of these observables are given in the next section. Here, we show how to install the python package and provide a list of functions within each class.

1.5.1 How to install CGMFtk

There are two ways of installing *CGMFtk*. First, pip can be used to install the python package locally. From the command line within the folder /cgmf/tools/, it can be installed with:

```
pip install .
```

(The pip version used depends on the python version that will be used with the package. It is recomended that python 3 or above be used.)

Pip is the recommended way to install *CGMFtk*, as the package will now be installed more globally on the user's machine and can easily be directly imported into a python script or *jupyter_notebook*.

Alternatively (or if pip is not available), the program can be install directly from the setup file:

```
python setup.py install
```

Again, python 3 and above is recommended.

This second command will install *CGMFtk* in the /site-packages folder of the corresponding python version directory. It is important to note that the user does not always have access to local directories, and in this case, the –user flag should be included:

```
python setup.py install --user
```

1.5.2 Classes within CGMFtk

The first class is the Histories class, which can be imported with

```
from CGMFtk import histories as fh
```

Note that if *CGMFtk* was installed using the python setup file, you will have to append the location of the package to your python path before importing the *histories* class,

```
import sys
sys.path.append('location-of-CGMFtk')
```

To load a **CGMF** history file into a history object

```
hist = fh.Histories('history.cgmf')
```

The second class is the Yields class which can be imported with

```
from CGMFtk import yields as yld
```

Loading a **CGMF** yield file is much the same as loading a history file

```
yields = yld.Yields('yields.cgmf')
```

Note that for both classes, you can provide a relative or an absolute path to the history file, if the file is not in the current directory.

1.5.3 The History Class

The functions within the fission history class are listed below.

1.5.4 The Yield Class

The functions within the yield class are listed below.

1.6 Examples of Jupyter Notebook

Jupyter notebooks offer very powerful and easy to use capabilities to analyze the raw output file of CGMF runs. Below are some examples of notebooks that analyze particular aspects of the prompt neutron and gamma data calculated by **CGMF**.

1.6.1 Reading CGMF Fission Yields

[1]:
```
### initializations and import libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
%pylab inline

from CGMFtk import yields as yld
```

```
Populating the interactive namespace from numpy and matplotlib
```

CGMF can be used to generate pre-neutron emission fission fragment yields in charge, mass, kinetic energy, excitation energy, spin, and parity. Those yields $Y(Z, A, KE, U, J, \pi)$ set the initial conditions for the decay of the fragments by emission of prompt neutrons and γ rays.

To create an output file containing such yields, CGMF can be used with a negative number of events using the '-n' option:

./cgmf.x -i 98252 -e 0.0 -n -500000

It creates an output file that can easily be read:

[2]:
```
yields = yld.Yields('98252-yields.cgmf')
```

The number of events in the file and number of fission fragments can be printed

[12]:
```
print ('Number of events: ',yields.numberEvents)
print ('Number of fission fragments: ',yields.numberFragments)
```

```
Number of events:  500000
Number of fission fragments:  1000000
```

Sorting the data by mass, charge, etc, is trivial:

[3]:
```
Z  = yields.getZ()
A  = yields.getZ()
KE = yields.getKE()
U  = yields.getU()
J  = yields.getJ()
p  = yields.getP()
```

The total kinetic energy and total excitation energy of each event is also easily available

[13]:
```
TKE = yields.getTKE()
TXE = yields.getTXE()
```

Manipulating and plotting those initial conditions can be done very simply with the help of numpy routines. For instance, finding the min, max, and mean values of the initial excitation energy in the fragments can be done by:

[4]:
```
print (np.min(U), np.max(U), np.mean(U))
```

```
0.0231779 62.9756 16.027517501660697
```

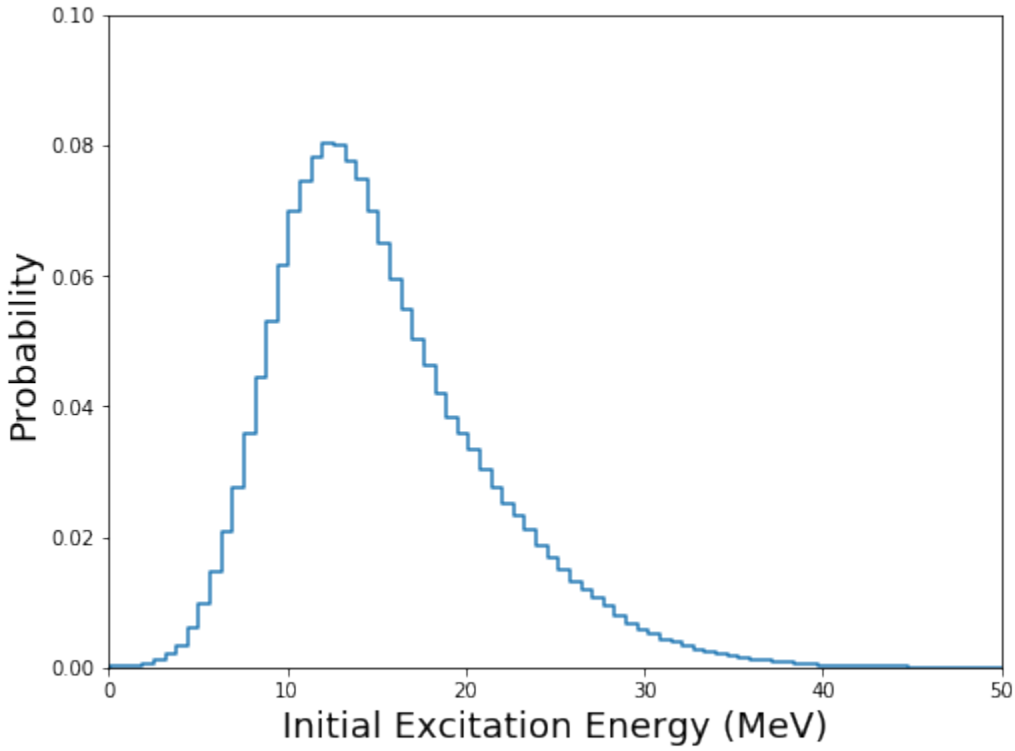
and plotting the distribution of the excitation energy:

[7]:
```
fig=figure(figsize(8,6))
h,b = np.histogram(U,bins=100,density=True)
```

(continues on next page)

(continued from previous page)

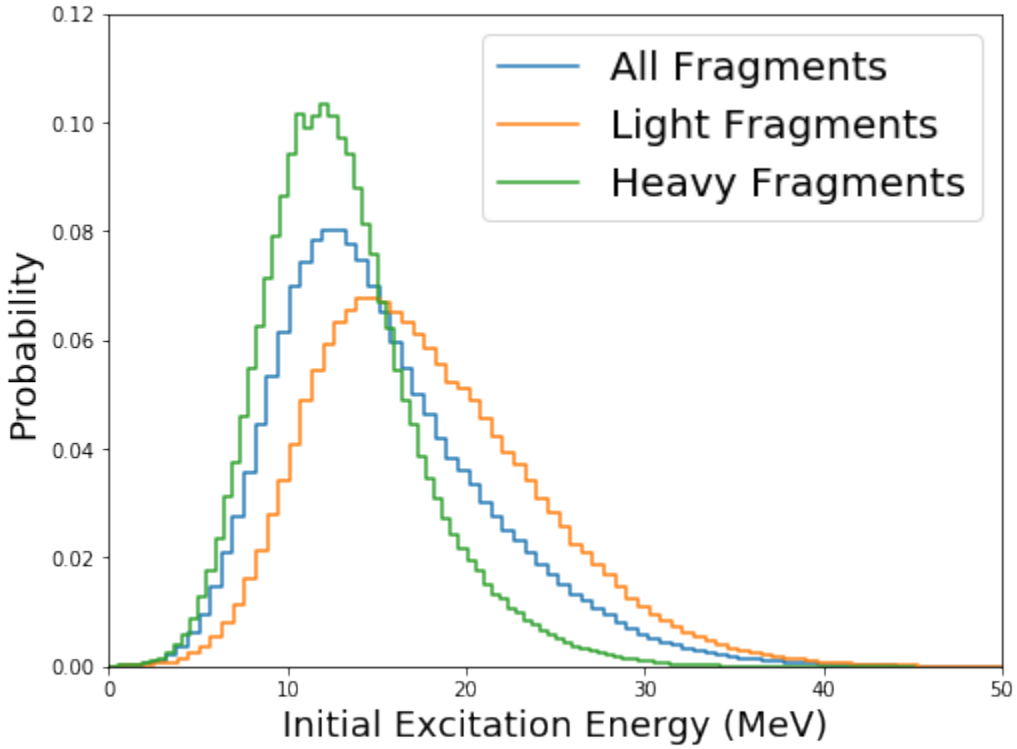
```
plt.step(b[:-1],h)
plt.xlim(0,50)
plt.xlabel("Initial Excitation Energy (MeV)",fontsize=18)
plt.ylim(0,0.1)
plt.ylabel("Probability",fontsize=18)
plt.show()
```



Separating the distributions for the light and heavy fragments can be done by recognizing that

[8]:
```
Ul = yields.getULF()  # light fragments
Uh = yields.getUHF() # heavy fragments
```

[11]:
```
fig=figure(figsize(8,6))
h,b = np.histogram(U,bins=100,density=True)
hl,bl = np.histogram(Ul,bins=100,density=True)
hh,bh = np.histogram(Uh,bins=100,density=True)
plt.step(b[:-1],h,label="All Fragments")
plt.step(bl[:-1],hl,label="Light Fragments")
plt.step(bh[:-1],hh,label="Heavy Fragments")
plt.xlim(0,50)
plt.xlabel("Initial Excitation Energy (MeV)",fontsize=18)
plt.ylim(0,0.12)
plt.ylabel("Probability",fontsize=18)
lg=plt.legend(fontsize=20)
plt.show()
```



Similar plots and statistical analyses can be carried out for any initial quantity.

A summary table can also be easily printed

[14]:
```
yields.summaryTable()
```

[14]:
```
[['', 'All Fragments', 'Light Fragments', 'Heavy Fragments'],
 ['A', '126.00', '108.62', '143.38'],
 ['Z', '49.00', '42.63', '55.37'],
 ['TXE / U (MeV)', '32.06', '18.52', '13.54'],
 ['TKE / KE (MeV)', '185.78', '105.57', '80.21'],
 ['J ($\\hbar$)', ' 9.34', ' 8.85', ' 9.83'],
 ['parity', ' 0.00', ' 0.00', ' 0.00']]
```

[ ]:

1.6.2 Reading CGMF Fission Events

[1]:
```
### initializations and import libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
%pylab inline

from CGMFtk import histories as fh
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
### rcParams are the default parameters for matplotlib
import matplotlib as mpl

print ("Matplotbib Version: ", mpl.__version__)

mpl.rcParams['font.size'] = 18
mpl.rcParams['font.family'] = 'Helvetica', 'serif'
#mpl.rcParams['font.color'] = 'darkred'
mpl.rcParams['font.weight'] = 'normal'

mpl.rcParams['axes.labelsize'] = 18.
mpl.rcParams['xtick.labelsize'] = 18.
mpl.rcParams['ytick.labelsize'] = 18.
mpl.rcParams['lines.linewidth'] = 2.

font = {'family' : 'serif',
        'color'  : 'darkred',
        'weight' : 'normal',
        'size'   : 18,
        }

mpl.rcParams['xtick.major.pad']='10'
mpl.rcParams['ytick.major.pad']='10'

mpl.rcParams['image.cmap'] = 'inferno'
```

```
Matplotbib Version:  3.1.3
```

The default output of a CGMF run is an ASCII file that contains characteristics of fission events. Once the **Histories** python class is uploaded, reading a CGMF output is straightforward:

[3]:
```
hist = fh.Histories ('98252sf.cgmf')
```

The number of fission events and fission fragments can be easily extracted from the **Histories** class

[4]:
```
print ('This file contains ',str(hist.getNumberEvents()),' events and ',str(hist.
↪getNumberFragments()),' fission fragments')
```

```
This file contains  500000  events and  1000000  fission fragments
```

With the option 'nevents', the number of fission events that are read can be specified:

hist = fh.Histories('92235_1MeV.cgmf',nevents=5000)

This can be particularly useful for testing routines on a small number of events since reading the full history file could be time consuming

Several python functions, part of the **Histories** class, make it easy to analyze those events.

Accessing Fission Histories

Although we rarely need to access directly the fission histories, it is possible by typing:

[5]:
```
events = hist.getFissionHistories()
```

[6]:
```
events[10]
```

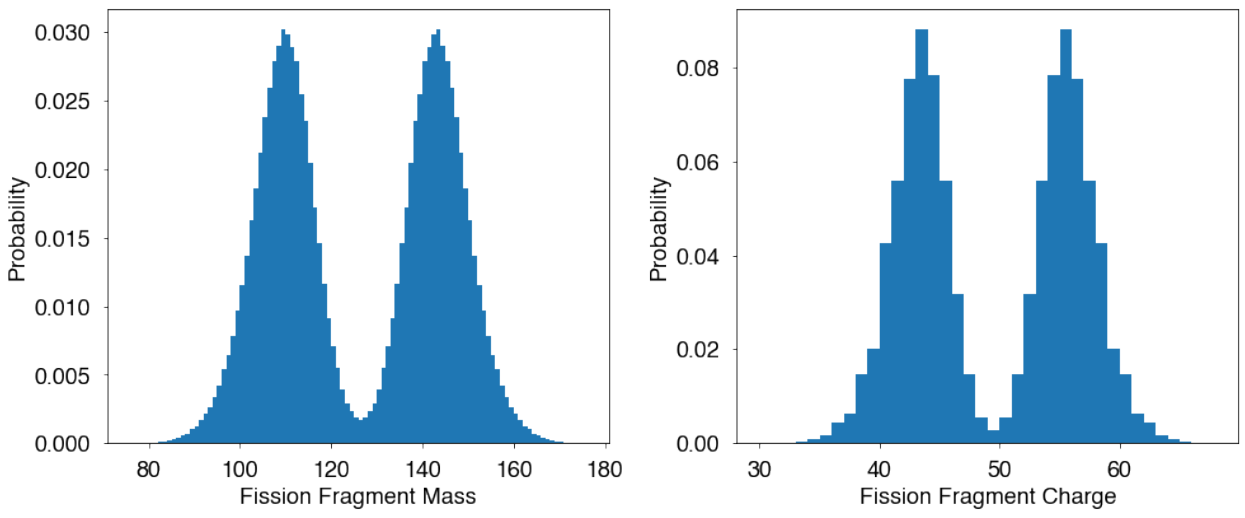
[6]:
```
array([115, 45, 17.218, 11.5, -1, 109.292, 2, 7, list([1.099, 1.435]),
       list([1.419, 4.427]),
       list([0.849, 0.848, 0.196, 0.071, 0.63, 0.231, 0.211]),
       list([0.849, 0.848, 0.196, 0.071, 0.63, 0.231, 0.211]),
       list([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]), 1036.646, -4397.075,
       -1729.17, 987.958, -4304.322, -1671.193,
       list([array([ 0.132,  0.663, -0.737]), array([ 0.473, -0.875,  0.107])]),
       list([array([ 0.292, -0.164, -0.942]), array([ 0.368, -0.924, -0.102])]),
       0, list([]), list([]), 104.141], dtype=object)
```

This history first list the characteristics of the fission fragment in mass, charge, excitation energy, spin, parity, and kinetic energy. It then provides the number of neutrons and gamma rays emitted in this event, followed by their characteristics in energy and direction.

Accessing Fission Fragment Characteristics

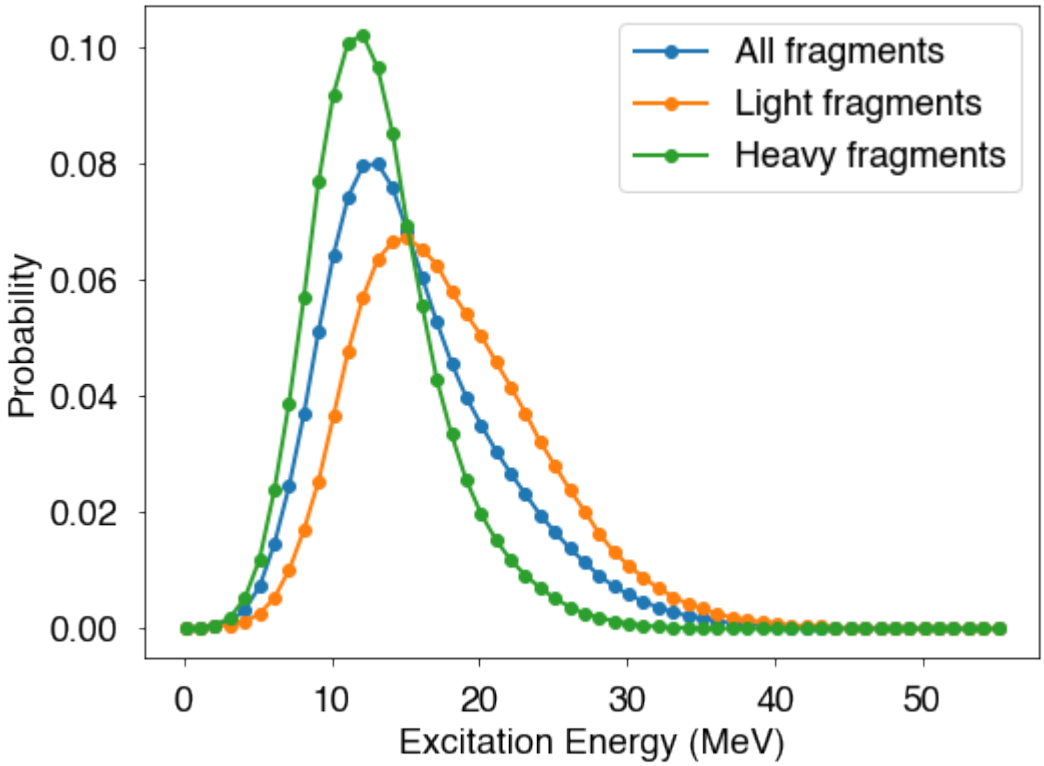
Fission fragment characteristics can be obtained using accessors such as: getA(), getZ(), getNu(), etc.

[8]:
```
A = hist.getA()
Z = hist.getZ()

fig=figure(figsize(14,6))
plt.subplot(1,2,1)
plt.hist(A,bins=np.arange(min(A),max(A)+1),density=True)
plt.xlabel("Fission Fragment Mass")
plt.ylabel("Probability")

plt.subplot(1,2,2)
plt.hist(Z,bins=np.arange(min(Z),max(Z)+1),density=True)
plt.xlabel("Fission Fragment Charge")
plt.ylabel("Probability")

plt.tight_layout()
plt.show()
```



Quantities for the light and heavy fragments can be extracted

[15]:
```
U = hist.getU()
Ul = hist.getULF()
Uh = hist.getUHF()
bx = np.arange(min(U),max(U)+1)

h,b = np.histogram(U,bins=bx,density=True)
hl,bl = np.histogram(Ul,bins=bx,density=True)
hh,bh = np.histogram(Uh,bins=bx,density=True)

fig = plt.figure(figsize=(8,6))
plt.plot(b[:-1],h,'o-',label='All fragments')
plt.plot(bl[:-1],hl,'o-',label='Light fragments')
plt.plot(bh[:-1],hh,'o-',label='Heavy fragments')
plt.legend()
plt.xlabel('Excitation Energy (MeV)')
plt.ylabel('Probability')
plt.show()
```



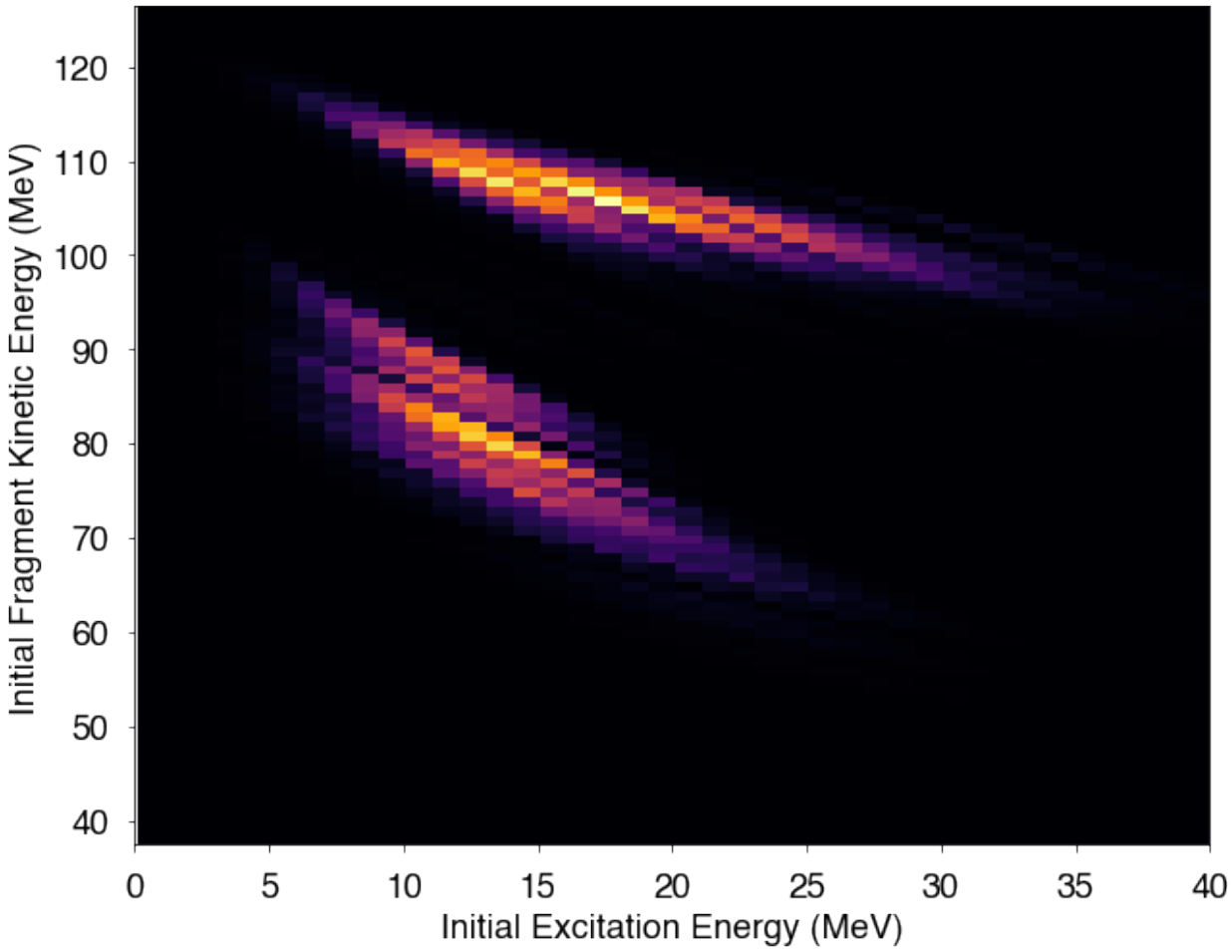
Plotting the initial excitation energy of the fragments as a function of its kinetic energy can be done by invoking getU() and getKEpre() (before neutron emission).

[12]:
```
U=hist.getU()
KE=hist.getKEpre()
bx=np.arange(min(U),max(U))
by=np.arange(min(KE),max(KE))

fig=figure(figsize(10,8))
plt.hist2d(U,KE,bins=(bx,by))
plt.xlim(0,40)
plt.xlabel("Initial Excitation Energy (MeV)")
```

(continues on next page)

(continued from previous page)

```
plt.ylabel("Initial Fragment Kinetic Energy (MeV)")
plt.show()
```



Summary table

A table summarizing the main characteristics of all fission events, fission fragments, neutrons and gamma rays can be generated by using the **summaryTable()** function.

[13]:
```
hist.summaryTable()
```

[13]:
```
[['',
  'All Fragments',
  'Light Fragments',
  'Heavy Fragments',
  'Pre-Fission',
  'Total'],
 ['A', '126.00', '108.61', '143.39'],
 ['Z', '49.00', '42.63', '55.37'],
 ['TXE / U (MeV)', '32.05', '18.51', '13.54'],
 ['TKE / KE (MeV)', '185.78', '105.57', '80.21'],
 ['J ($\\hbar$)', ' 9.35', ' 8.87', ' 9.84'],
 ['parity', ' 0.00', ' 0.00', ' 0.00'],
```

(continues on next page)

(continued from previous page)

```
 ['$\\langle \\nu\\rangle$', '3.818', '2.136', '1.681', '0.000', '3.818'],
 ['$\\langle \\epsilon_n^{cm}\\rangle$ (MeV)', '1.303', '1.372', '1.214'],
 ['$\\langle E_n^{lab}\\rangle$ (MeV)',
  '2.077',
  '2.329',
  '1.757',
  '0.000',
  '2.077'],
 ['$\\langle \\nu_\\gamma\\rangle$', ' 8.32', ' 4.29', ' 4.03'],
 ['$\\langle E_\\gamma^{lab}\\rangle$ (MeV)', ' 0.75', ' 0.76', ' 0.75']]
```

[ ]:

1.6.3 Analyzing Prompt Neutrons

[1]:
```
### initializations and import libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
%pylab inline

from CGMFtk import histories as fh
```

```
Populating the interactive namespace from numpy and matplotlib
```

[16]:
```
### rcParams are the default parameters for matplotlib
import matplotlib as mpl

print ("Matplotbib Version: ", mpl.__version__)

mpl.rcParams['font.size'] = 18
mpl.rcParams['font.family'] = 'Helvetica', 'serif'
#mpl.rcParams['font.color'] = 'darkred'
mpl.rcParams['font.weight'] = 'normal'

mpl.rcParams['axes.labelsize'] = 18.
mpl.rcParams['xtick.labelsize'] = 18.
mpl.rcParams['ytick.labelsize'] = 18.
mpl.rcParams['lines.linewidth'] = 2.

font = {'family' : 'serif',
        'color'  : 'darkred',
        'weight' : 'normal',
        'size'   : 18,
        }

mpl.rcParams['xtick.major.pad']='10'
mpl.rcParams['ytick.major.pad']='10'

mpl.rcParams['image.cmap'] = 'inferno'
```

```
Matplotbib Version:  3.1.3
```

First we load in the history file

[2]:
```
hist = fh.Histories('98252sf.cgmf')
```

Many average quantities for the pre-fission and prompt neutrons can be calculated using the histories class

[3]:
```
# the average neutron multiplicity
print ('nubar (per fission event) = ',hist.nubartot())
print ('average number of neutrons per fragment = ',hist.nubar())
```

```
nubar (per fission event) =  3.817616
average number of neutrons per fragment =  1.908808
```

Or we might want a list of the multiplicity of each event

[7]:
```
nu = hist.getNutot()
print (nu[:10])
# hist.getNu() pulls the multiplicity for each fission fragment
# hist.getNuEvent() pulls the multiplicity for each event, not including pre-fission␣
↪neutrons
```

```
[4 3 6 5 4 3 3 3 4 4]
```

Neutron energies in both the center of mass (of the compound nucleus) and the laboratory frame are available

[8]:
```
Elab = hist.getNeutronElab()
Ecm = hist.getNeutronEcm()
```

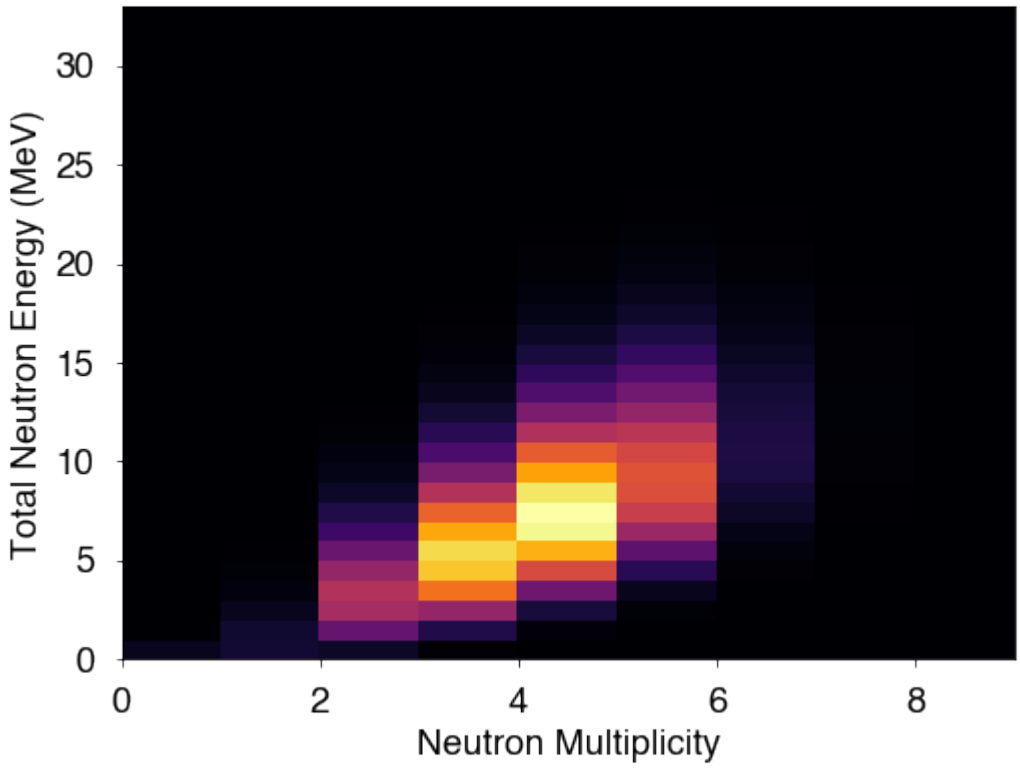
These two functions provide a list of lists of neutron energies for each fragment. Typically, more manipulation will be required to use these values for plotting. For example, if we want to plot the neutron multiplicity and the total neutron energy of each event:

[13]:
```
Elabtot = []
ElabEvent = Elab[::2]+Elab[1::2]
for x in ElabEvent:
    Elabtot.append(np.sum(x))
```

[14]:
```
print (len(nu))
print (len(Elabtot))
```

```
500000
500000
```

[17]:
```
fig = plt.figure(figsize=(8,6))
bx = np.arange(min(nu),max(nu)+1)
by = np.arange(min(Elabtot),max(Elabtot)+1)

plt.hist2d(nu,Elabtot,bins=(bx,by))
plt.xlabel('Neutron Multiplicity')
plt.ylabel('Total Neutron Energy (MeV)')
plt.show()
```



There are also routines to calculate the neutron average energies

[19]:
```
# average neutron energies
print ('Neutron energies in the lab:')
print ('Average energy of all neutrons = ',hist.meanNeutronElab())
print ('Average energy of neutrons from fragments = ',hist.meanNeutronElabFragments())
print ('Average energy of neutrons from light fragment = ',hist.meanNeutronElabLF())
print ('Average energy of neutrons from heavy fragment = ',hist.meanNeutronElabHF())
print (' ')
print ('Neutron energies in the center of mass:')
print ('Average energy of neutrons from fragments = ',hist.meanNeutronEcmFragments())
print ('Average energy of neutrons from light fragment = ',hist.meanNeutronEcmLF())
print ('Average energy of neutrons from heavy fragment = ',hist.meanNeutronEcmHF())
```

```
Neutron energies in the lab:
Average energy of all neutrons =  2.077427439009056
Average energy of neutrons from fragments =  2.077427439009056
Average energy of neutrons from light fragment =  2.3293141780172957
Average energy of neutrons from heavy fragment =  1.7574275095987781

Neutron energies in the center of mass:
Average energy of neutrons from fragments =  1.3025117701727997
Average energy of neutrons from light fragment =  1.3719594663675614
Average energy of neutrons from heavy fragment =  1.2142845853375386
```

[ ]:

1.6.4 Prompt Fission Neutron Spectrum

[1]:
```
### initializations and import libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
%pylab inline

from CGMFtk import histories as fh
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
### rcParams are the default parameters for matplotlib
import matplotlib as mpl

print ("Matplotbib Version: ", mpl.__version__)

mpl.rcParams['font.size'] = 18
mpl.rcParams['font.family'] = 'Helvetica', 'serif'
#mpl.rcParams['font.color'] = 'darkred'
mpl.rcParams['font.weight'] = 'normal'

mpl.rcParams['axes.labelsize'] = 18.
mpl.rcParams['xtick.labelsize'] = 18.
mpl.rcParams['ytick.labelsize'] = 18.
mpl.rcParams['lines.linewidth'] = 2.

font = {'family' : 'serif',
        'color'  : 'darkred',
        'weight' : 'normal',
        'size'   : 18,
        }

mpl.rcParams['xtick.major.pad']='10'
mpl.rcParams['ytick.major.pad']='10'

mpl.rcParams['image.cmap'] = 'inferno'
```

```
Matplotbib Version:  3.1.3
```

First, we read the default CGMF output.

[3]:
```
hist = fh.Histories('98252sf.cgmf')
```

Neutron energies in the center-of-mass and laboratory reference frames can be obtained as:

[5]:
```
Ecm  = hist.getNeutronEcm()
Elab = hist.getNeutronElab()
```

Extracting the list of neutron energies in the center-of-mass frame of the emitting fragment for the fission fragment number 154 (light fragment of the 76th event):

[8]:
```
print (Ecm[154])
```

```
[6.987, 0.665, 0.88]
```

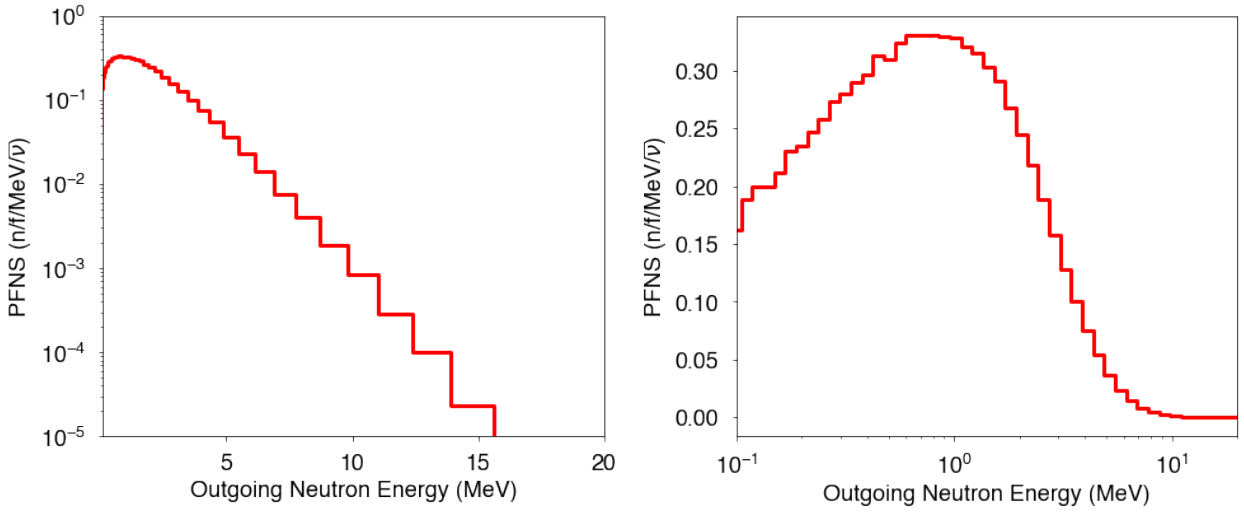
All neutron energies can then be binned in histograms, and analyzed and plotted that way. The `CGMF` python package **Histories** come with a function to directly extract PFNS:

[9]:
```
eout,pfns = hist.pfns()
```

which returns two arrays: (1) the outgoing energy grid (midpoints in MeV); (2) the prompt fission neutron spectrum (in n/MeV/nu-bar).

The result can be plotted using:

[10]:
```
fig=figure(figsize(14,6))

plt.subplot(1,2,1)
plt.step(eout,pfns,'r-',linewidth=3,where='mid')
plt.xlim(0.1,20.0)
plt.ylim(1e-5,1.0)
plt.xlabel("Outgoing Neutron Energy (MeV)")
plt.ylabel(r"PFNS (n/f/MeV/$\overline{\nu}$)")
plt.yscale('log')

plt.subplot(1,2,2)
plt.step(eout,pfns,'r-',linewidth=3,where='mid')
plt.xlim(0.1,20.0)
plt.xscale('log')
plt.xlabel("Outgoing Neutron Energy (MeV)")
plt.ylabel(r"PFNS (n/f/MeV/$\overline{\nu}$)")

plt.tight_layout()
plt.show()
```



[ ]:

1.6.5 Analyzing Prompt Gamma Rays

[1]:
```
### initializations and import libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
%pylab inline
```

(continues on next page)

(continued from previous page)

```
from CGMFtk import histories as fh
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
### rcParams are the default parameters for matplotlib
import matplotlib as mpl

print ("Matplotbib Version: ", mpl.__version__)

mpl.rcParams['font.size'] = 18
mpl.rcParams['font.family'] = 'Helvetica', 'serif'
#mpl.rcParams['font.color'] = 'darkred'
mpl.rcParams['font.weight'] = 'normal'

mpl.rcParams['axes.labelsize'] = 18.
mpl.rcParams['xtick.labelsize'] = 18.
mpl.rcParams['ytick.labelsize'] = 18.
mpl.rcParams['lines.linewidth'] = 2.

font = {'family' : 'serif',
        'color'  : 'darkred',
        'weight' : 'normal',
        'size'   : 18,
        }

mpl.rcParams['xtick.major.pad']='10'
mpl.rcParams['ytick.major.pad']='10'

mpl.rcParams['image.cmap'] = 'inferno'
```

```
Matplotbib Version:  3.1.3
```

First, we load the history file

[3]:
```
hist = fh.Histories('98252sf.cgmf')
```

```
98252sf.cgmf
```

Prompt gamma rays can be studied similarly to neutrons, although additional routines can be used to study gamma emissions between discrete states in specific fission fragments. Many of the routines in **CGMFtk** will look very similar to those used to construct neutron properties.

[4]:
```
# the average gamma multiplicity
print ('nubarg (per fission event) = ',hist.nubargtot())
print ('average number of gammas per fragment = ',hist.nubarg())
```

```
nubarg (per fission event) =  8.32369
average number of gammas per fragment =  4.161845
```

We can list the multiplicity of each event

[6]:
```
nug = hist.getNugtot()
print (nug[:10])

# hist.getNug() pulls the multiplicity for each fission fragment
```

```
[11  6  7  4  5 12 13 10 12  9]
```

Energies are only available in the lab frame - they are only doppler corrected from the center of mass

As with the neutrons, this function provides a list of energies for each fragment, not each event

[8]:
```
Elab = hist.getGammaElab()

# to manipulate the list to provide a list of energies for each event
ElabEvent = Elab[::2] + Elab[1::2]
```

In addition, there are routines to calculate the average gamma energies as well

[9]:
```
# average neutron energies
print ('Gamma energies in the lab:')
print ('Average energy of all gammas = ',hist.meanGammaElab())
print ('Average energy of gammas from light fragment = ',hist.meanGammaElabLF())
print ('Average energy of gammas from heavy fragment = ',hist.meanGammaElabHF())
```

```
Gamma energies in the lab:
Average energy of all gammas =  0.75293912339359
Average energy of gammas from light fragment =  0.7589534545393586
Average energy of gammas from heavy fragment =  0.7465280697469625
```

Finally, we also include a routine to perform gamma spectroscopy, using the discrete levels from RIPL. It is important to note that **CGMF** does not predict the discrete levels or branching ratios. Instead, **CGMF** computes the neutron and γ emission prior to feeding the discrete states.

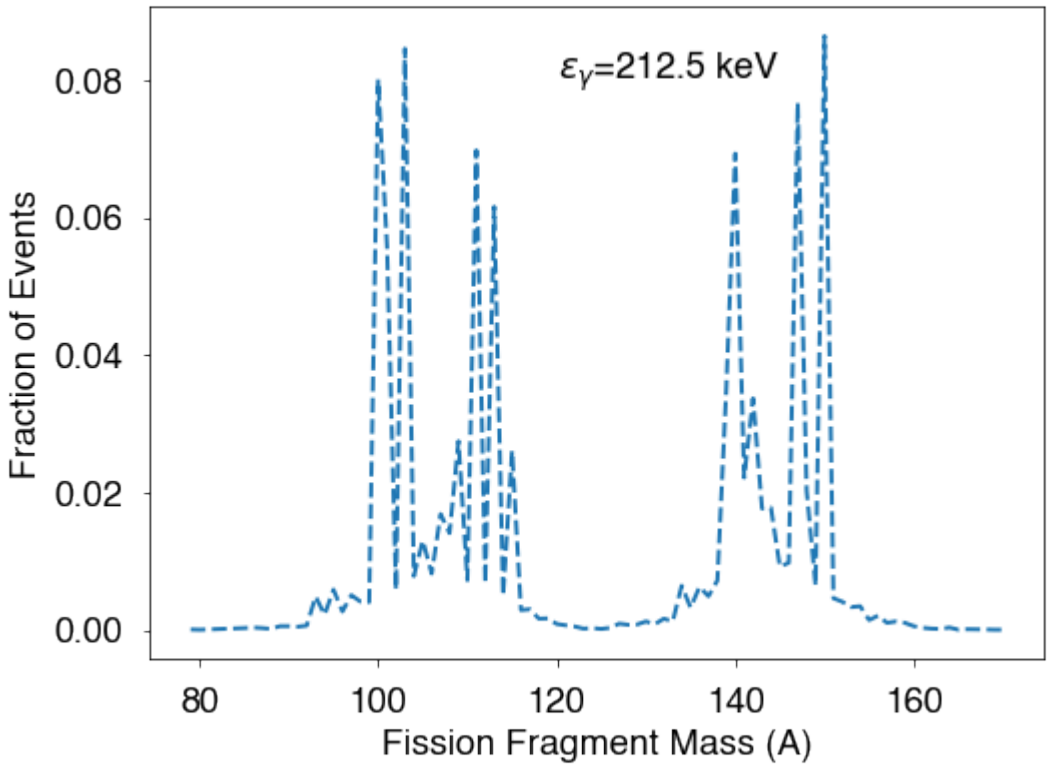
To run the `gammaSpec` function, the γ ray energy (in MeV) needs to be given, and the width of the energy bin around the given γ ray. Including `post=True` means that the masses and charges of the fragments after neutron emission will be returned.

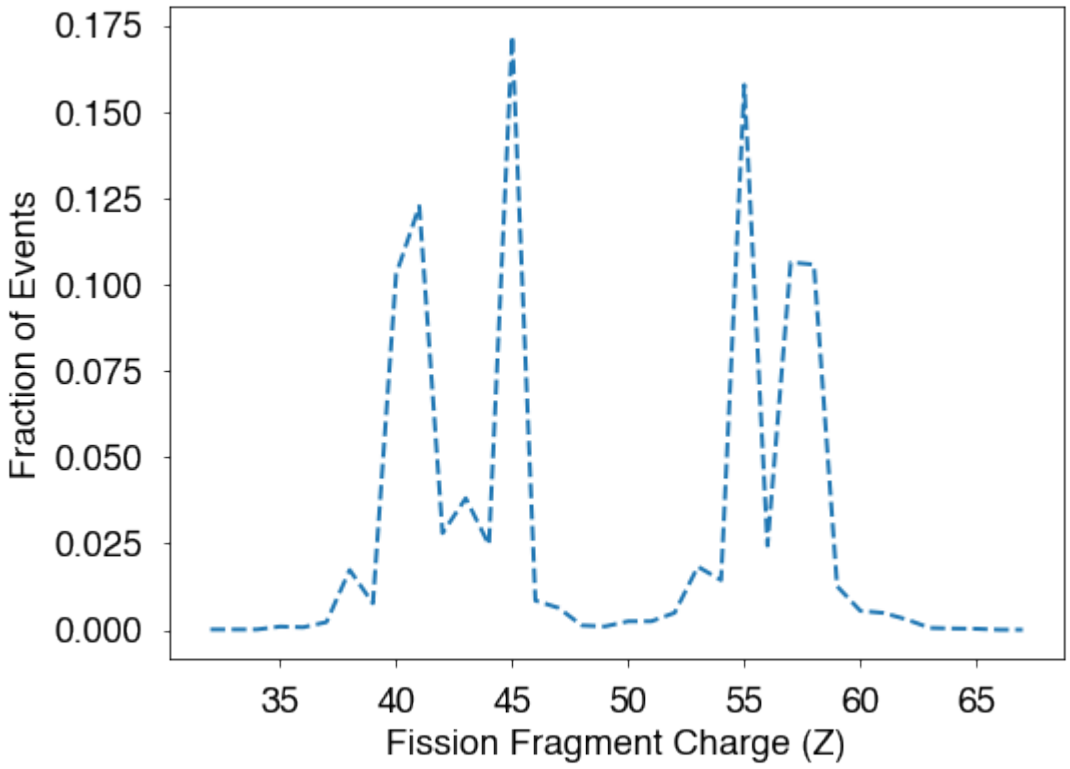
[12]:
```
# perform gamma-ray spectroscopy
gE = 0.2125 # gamma ray at 212.5 keV
dE = 0.01 # 1% energy resolution
gspec1 = hist.gammaSpec(gE,dE*gE,post=True)

# calculate the percentage of events for each A/Z
As1 = np.unique(gspec1[:,1])
totEvents = len(gspec1)
fracA1 = []
for A in As1:
    mask = gspec1[:,1]==A
    fracA1.append(len(gspec1[mask])/totEvents)
Zs1 = np.unique(gspec1[:,0])
fracZ1 = []
for Z in Zs1:
    mask = gspec1[:,0]==Z
    fracZ1.append(len(gspec1[mask])/totEvents)

fig = plt.figure(figsize=(8,6))
plt.plot(As1,fracA1,'--')
plt.xlabel('Fission Fragment Mass (A)')
plt.ylabel('Fraction of Events')
plt.text(120,0.08,r'$\epsilon_\gamma$=212.5 keV',fontsize=18)
plt.show()

fig = plt.figure(figsize=(8,6))
plt.plot(Zs1,fracZ1,'--')
plt.xlabel('Fission Fragment Charge (Z)')
```

(continues on next page)

(continued from previous page)

```
plt.ylabel('Fraction of Events')
plt.show()
```





[ ]:

1.6.6 Prompt Fission Gamma Spectrum

[1]:
```
### initializations and import libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
%pylab inline

from CGMFtk import histories as fh
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
### rcParams are the default parameters for matplotlib
import matplotlib as mpl

print ("Matplotbib Version: ", mpl.__version__)

mpl.rcParams['font.size'] = 18
mpl.rcParams['font.family'] = 'Helvetica', 'serif'
#mpl.rcParams['font.color'] = 'darkred'
mpl.rcParams['font.weight'] = 'normal'

mpl.rcParams['axes.labelsize'] = 18.
mpl.rcParams['xtick.labelsize'] = 18.
mpl.rcParams['ytick.labelsize'] = 18.
mpl.rcParams['lines.linewidth'] = 2.

font = {'family' : 'serif',
        'color'  : 'darkred',
        'weight' : 'normal',
        'size'   : 18,
        }

mpl.rcParams['xtick.major.pad']='10'
mpl.rcParams['ytick.major.pad']='10'

mpl.rcParams['image.cmap'] = 'inferno'
```

```
Matplotbib Version:  3.1.3
```

First, we load the history file

[3]:
```
hist = fh.Histories('98252sf.cgmf')
```

```
98252sf.cgmf
```

Only gamma-ray energies in the laboratory frame are saved in the history files

[4]:
```
Elab = hist.getGammaElab()
```

As for the neutrons, we can pull out the list of energies from individual fragments

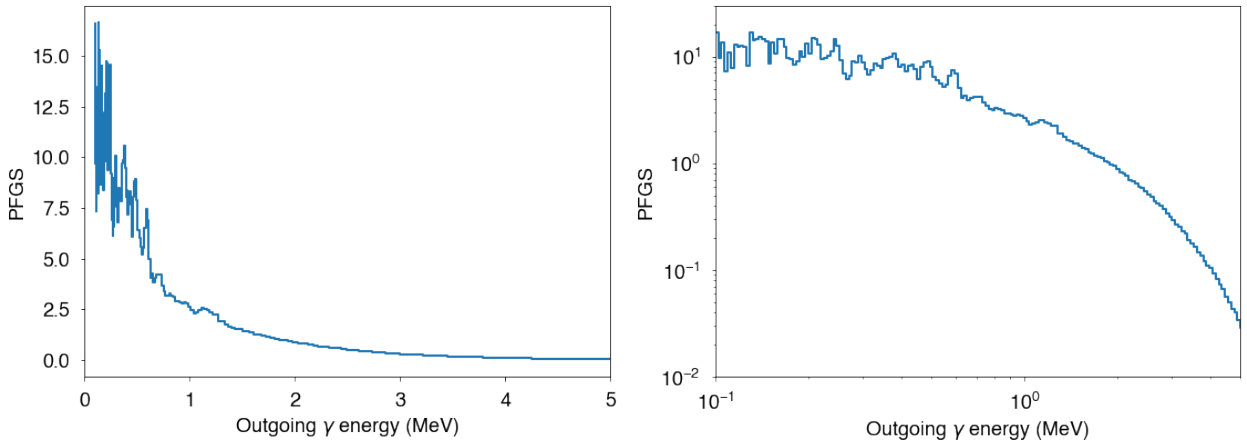
[5]:
```
print (Elab[5])
```

```
[1.075, 0.819, 1.433]
```

The PFGS can be easily calculated with `CGMFtk` and plotted

[6]:
```
ebins,pfgs = hist.pfgs()
```

[9]:
```
# construct and plot the prompt gamma spectrum
fig = plt.figure(figsize=(16,6))
plt.subplot(121)
ebins,pfgs = hist.pfgs()
plt.step(ebins,pfgs,where='mid')
plt.xlim(0,5)
plt.xlabel(r'Outgoing $\gamma$ energy (MeV)')
plt.ylabel('PFGS')
plt.subplot(122)
plt.step(ebins,pfgs,where='mid')
plt.xlim(0.1,5)
plt.xscale('log')
plt.yscale('log')
plt.xlabel(r'Outgoing $\gamma$ energy (MeV)')
plt.ylabel('PFGS')
plt.ylim(1e-2,30)
plt.tight_layout()
plt.show()
```



[ ]:

1.6.7 Extracting Neutron and Photon Multiplicity Distributions

Last version: Oct. 29, 2020

Prompt neutron and photon multiplicity distributions can be easily extracted from fission event files created by the CGMF code. Below we show an example of built-in functions provided in the **CGMFtk** Python package, as well as manual manipulations of the event file to reconstruct the same results.

Also, the importance of defining the detector energy threshold and time coincidence window around the fission event is also discussed.

[1]:
```
### initializations and import libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
%pylab inline

from CGMFtk import histories as fh
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
### rcParams are the default parameters for matplotlib
import matplotlib as mpl

print ("Matplotbib Version: ", mpl.__version__)

mpl.rcParams['font.size'] = 18
mpl.rcParams['font.family'] = 'Helvetica', 'serif'
#mpl.rcParams['font.color'] = 'darkred'
mpl.rcParams['font.weight'] = 'normal'

mpl.rcParams['axes.labelsize'] = 18.
mpl.rcParams['xtick.labelsize'] = 18.
mpl.rcParams['ytick.labelsize'] = 18.
mpl.rcParams['lines.linewidth'] = 2.

font = {'family' : 'serif',
        'color'  : 'darkred',
        'weight' : 'normal',
        'size'   : 18,
        }

mpl.rcParams['xtick.major.pad']='10'
mpl.rcParams['ytick.major.pad']='10'

mpl.rcParams['image.cmap'] = 'inferno'
```

```
Matplotbib Version:  3.1.3
```

Reading fission events file

[3]:
```
hist = fh.Histories('98252sf.cgmf')
```

```
98252sf.cgmf
```

h now contains all fission events, as calculated by CGMF in this particular example file.

Inferring $P(\nu)$ and $P(N_\gamma)$

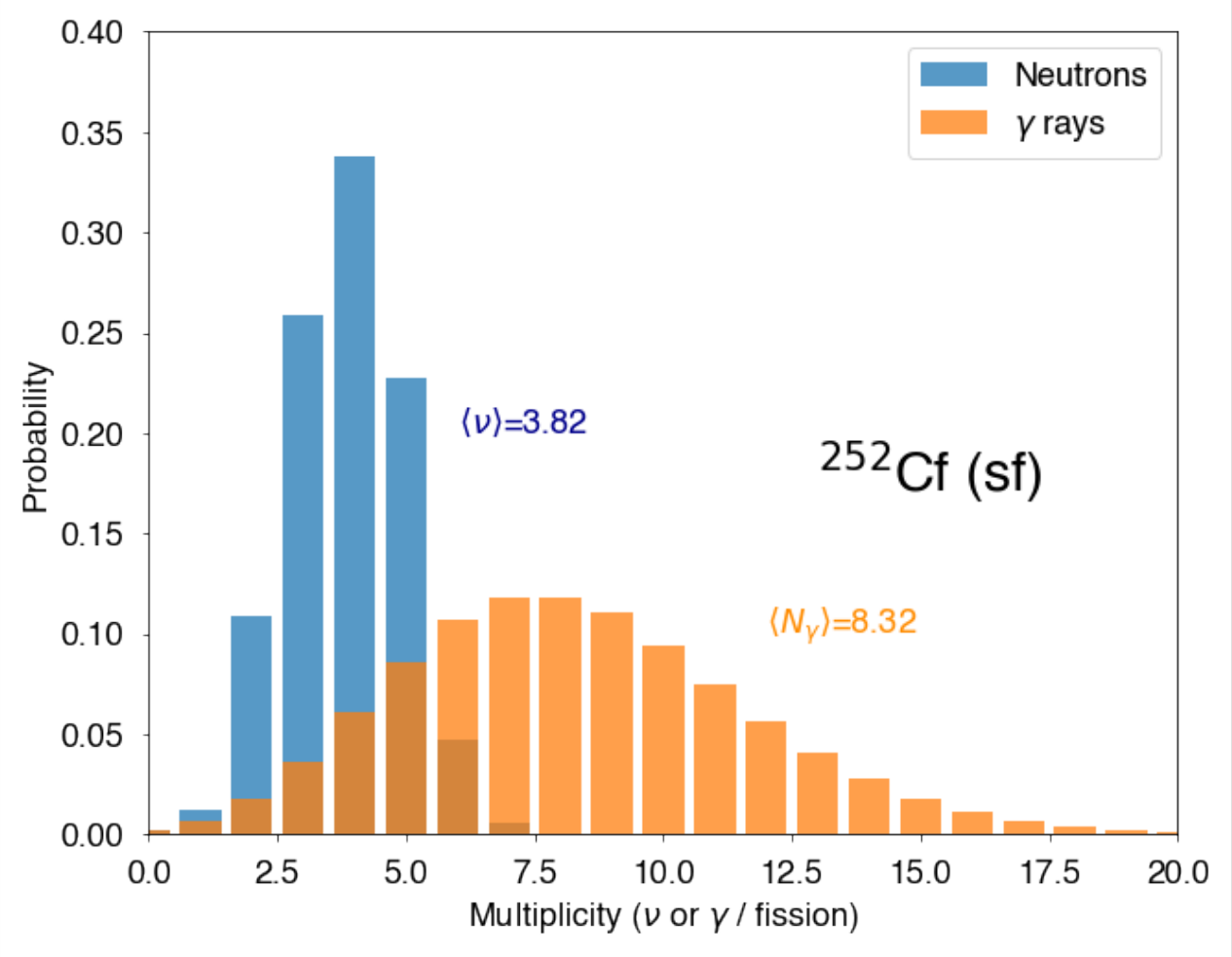
The neutron and γ-ray multiplicity distributions can be easily accessed by invoking built-in functions from the **CGMFtk** package:

[4]:
```
print ("Neutrons: ", hist.Pnu())
print ("Gamma Rays: ", hist.Pnug())
```

```
Neutrons:  (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), array([2.33200e-03, 1.20720e-02, 1.
↪08506e-01, 2.58974e-01, 3.37622e-01,
       2.27606e-01, 4.73280e-02, 5.36000e-03, 1.96000e-04, 4.00000e-06]))
Gamma Rays:  (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,␣
↪16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]), array([1.
↪66800e-03, 6.34400e-03, 1.72960e-02, 3.58580e-02, 6.05880e-02,
       8.58900e-02, 1.07336e-01, 1.18104e-01, 1.18238e-01, 1.10204e-01,
       9.37640e-02, 7.47020e-02, 5.63660e-02, 4.11320e-02, 2.76920e-02,
       1.76760e-02, 1.08940e-02, 6.93600e-03, 4.04200e-03, 2.31800e-03,
       1.34800e-03, 7.40000e-04, 3.68000e-04, 2.54000e-04, 1.08000e-04,
       5.20000e-05, 2.80000e-05, 2.20000e-05, 1.40000e-05, 1.00000e-05,
       2.00000e-06, 4.00000e-06, 2.00000e-06]))
```

and plotting them using:

[7]:
```
fig=figure(figsize(10,8))

ax=plt.subplot(1,1,1)

nu,Pnu = hist.Pnu()
plt.bar(nu,Pnu,label="Neutrons",alpha=0.75)
plt.text(6,0.2,r"$\langle \nu \rangle$={0:4.2f}".format(hist.nubar()*2),color=
↪"darkblue")

nug,Pnug = hist.Pnug()
plt.bar(nug,Pnug,label=r"$\gamma$ rays",alpha=0.75)
plt.text(12,0.1,r"$\langle N_\gamma \rangle$={0:4.2f}".format(hist.nubarg()*2),color=
↪"darkorange")

plt.xlim(0,20)
plt.xlabel(r"Multiplicity ($\nu$ or $\gamma$ / fission)")
plt.ylim(0,0.4)
plt.ylabel("Probability")

plt.text(13,0.17,r"$^{252}$Cf (sf)",fontsize=30)
lg=plt.legend()

plt.show()
```



In that case, we also used the functions to calculate the mean values of the neutron and γ multiplicity distributions respectively, `hist.nubar()` and `hist.nubarg()`. Note that those functions return the mean values for each fission fragment decay, so they need to be multiplied by two to get the mean values for a fission event, with two fission fragments. The neutron and γ multiplicity averages for each event can be calculated with `hist.nubartot()` and `hist.nubargtot()`.

Manually

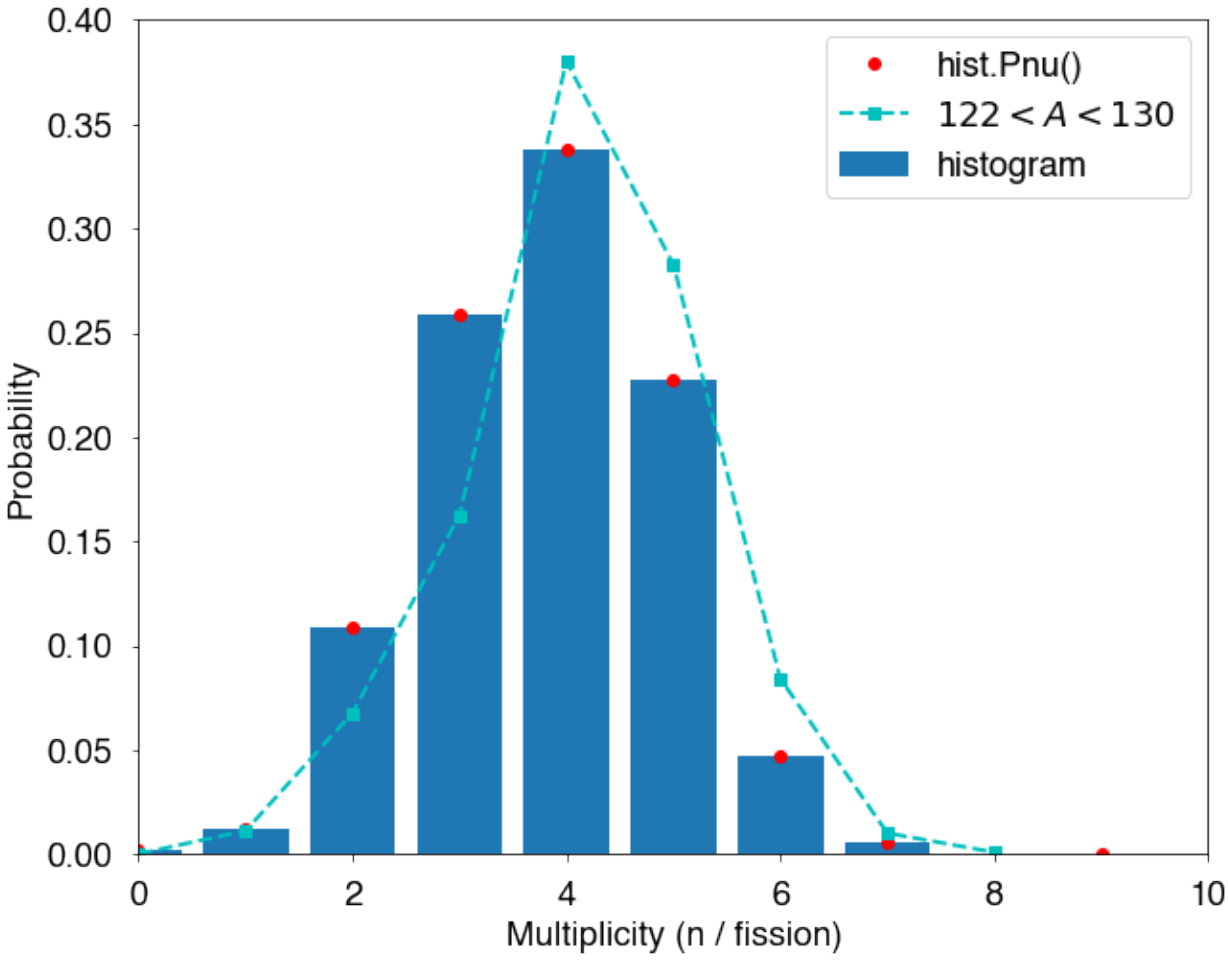
The same information can of course be extracted by direct analysis of the fission events, as follows:

[14]:
```
#-- Building an histogram of the sum of the neutron multiplicities for
#-- both light and heavy fragments

nu,Pnu = hist.Pnu()
data=np.histogram(hist.getNu()[::2]+hist.getNu()[1::2],bins=nu,density=True)

#-- The advantage is that one can easily apply some masks to the histograms
#-- For instance, counting only neutrons from fragments with mass greater than 130
mask=np.logical_and(hist.getA()>122,hist.getA()<130)
hmasked=hist.getNu()[mask]
datamasked=np.histogram(hmasked[::2]+hmasked[1::2],bins=nu,density=True)
```

(continues on next page)

(continued from previous page)

```
fig=figure(figsize(10,8))

ax=plt.subplot(1,1,1)

plt.bar(nu[:-1],data[0][:],label="histogram")
plt.plot(nu,Pnu,'ro',label="hist.Pnu()")
plt.plot(nu[:-1],datamasked[0][:],'cs--',label="$122<A<130$")

plt.xlim(0,10)
plt.xlabel("Multiplicity (n / fission)")
plt.ylim(0,0.4)
plt.ylabel("Probability")

plt.legend()
plt.show()
```

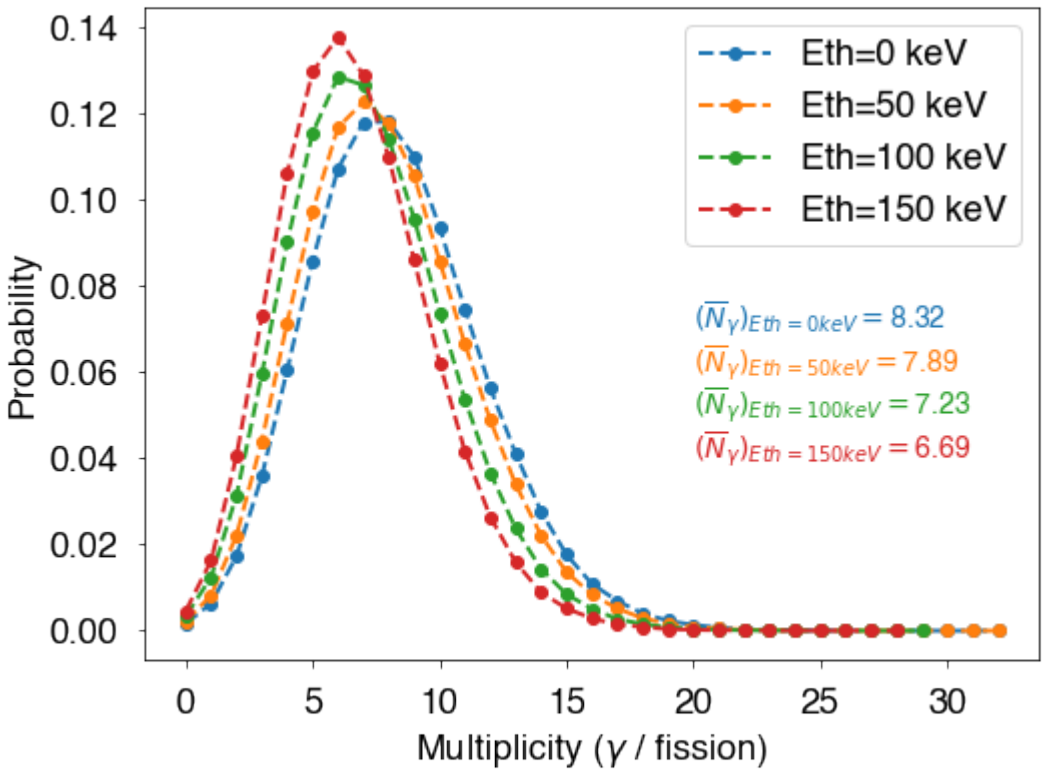


The average γ-ray multiplicity is very sensitive to the detection energy threshold as well as the time coincidence window with the fission event.

γ-ray timinig is discussed in another example. However, several of the methods in **CGMFtk.Histories** already contain easy ways to take the γ-ray threshold energy into account

[18]:
```
nug,Pnug = hist.Pnug()
nug005,Pnug005 = hist.Pnug(Eth=0.05) # 50 keV gamma-ray threshold energy
nug010,Pnug010 = hist.Pnug(Eth=0.1) # 100 keV gamma-ray threshold energy
nug015,Pnug015 = hist.Pnug(Eth=0.15) # 150 keV gamma-ray threshold energy
```

[24]:
```
# calculate the average gamma multiplicity with each energy threshold
Ng0 = np.sum(nug*Pnug)
Ng005 = np.sum(nug005*Pnug005)
Ng010 = np.sum(nug010*Pnug010)
Ng015 = np.sum(nug015*Pnug015)

fig = plt.figure(figsize=(8,6))
plt.plot(nug,Pnug,'o--',label='Eth=0 keV')
plt.plot(nug005,Pnug005,'o--',label='Eth=50 keV')
plt.plot(nug010,Pnug010,'o--',label='Eth=100 keV')
plt.plot(nug015,Pnug015,'o--',label='Eth=150 keV')
plt.text(20,0.07,r'$(\overline{N}_\gamma)_{Eth=0 keV}=$'+str(round(Ng0,2)),
↪fontsize=14,color='C0')
plt.text(20,0.06,r'$(\overline{N}_\gamma)_{Eth=50 keV}=$'+str(round(Ng005,2)),
↪fontsize=14,color='C1')
plt.text(20,0.05,r'$(\overline{N}_\gamma)_{Eth=100 keV}=$'+str(round(Ng010,2)),
↪fontsize=14,color='C2')
plt.text(20,0.04,r'$(\overline{N}_\gamma)_{Eth=150 keV}=$'+str(round(Ng015,2)),
↪fontsize=14,color='C3')
plt.xlabel('Multiplicity ($\gamma$ / fission)')
plt.ylabel('Probability')
plt.legend()
plt.show()
```



[ ]:

1.6.8 Prompt Particle Correlations

Last version: Oct. 28, 2020

Many correlations exist between the emitted prompt neutrons and photons and the fission fragments that evaporated them. In this notebook, we demonstrate how to analyze a few of those correlations.

[1]:
```
### initializations and import libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
%pylab inline

from CGMFtk import histories as fh
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
### rcParams are the default parameters for matplotlib
import matplotlib as mpl

print ("Matplotbib Version: ", mpl.__version__)

mpl.rcParams['font.size'] = 18
mpl.rcParams['font.family'] = 'Helvetica', 'serif'
#mpl.rcParams['font.color'] = 'darkred'
mpl.rcParams['font.weight'] = 'normal'

mpl.rcParams['axes.labelsize'] = 18.
mpl.rcParams['xtick.labelsize'] = 18.
mpl.rcParams['ytick.labelsize'] = 18.
mpl.rcParams['lines.linewidth'] = 2.

font = {'family' : 'serif',
        'color'  : 'darkred',
        'weight' : 'normal',
        'size'   : 18,
        }

mpl.rcParams['xtick.major.pad']='10'
mpl.rcParams['ytick.major.pad']='10'

mpl.rcParams['image.cmap'] = 'inferno'
```

```
Matplotbib Version:  3.1.3
```

Reading fission events file

First, we again load the history file

[3]:
```
hist = fh.Histories ('98252sf.cgmf')
```

```
98252sf.cgmf
```

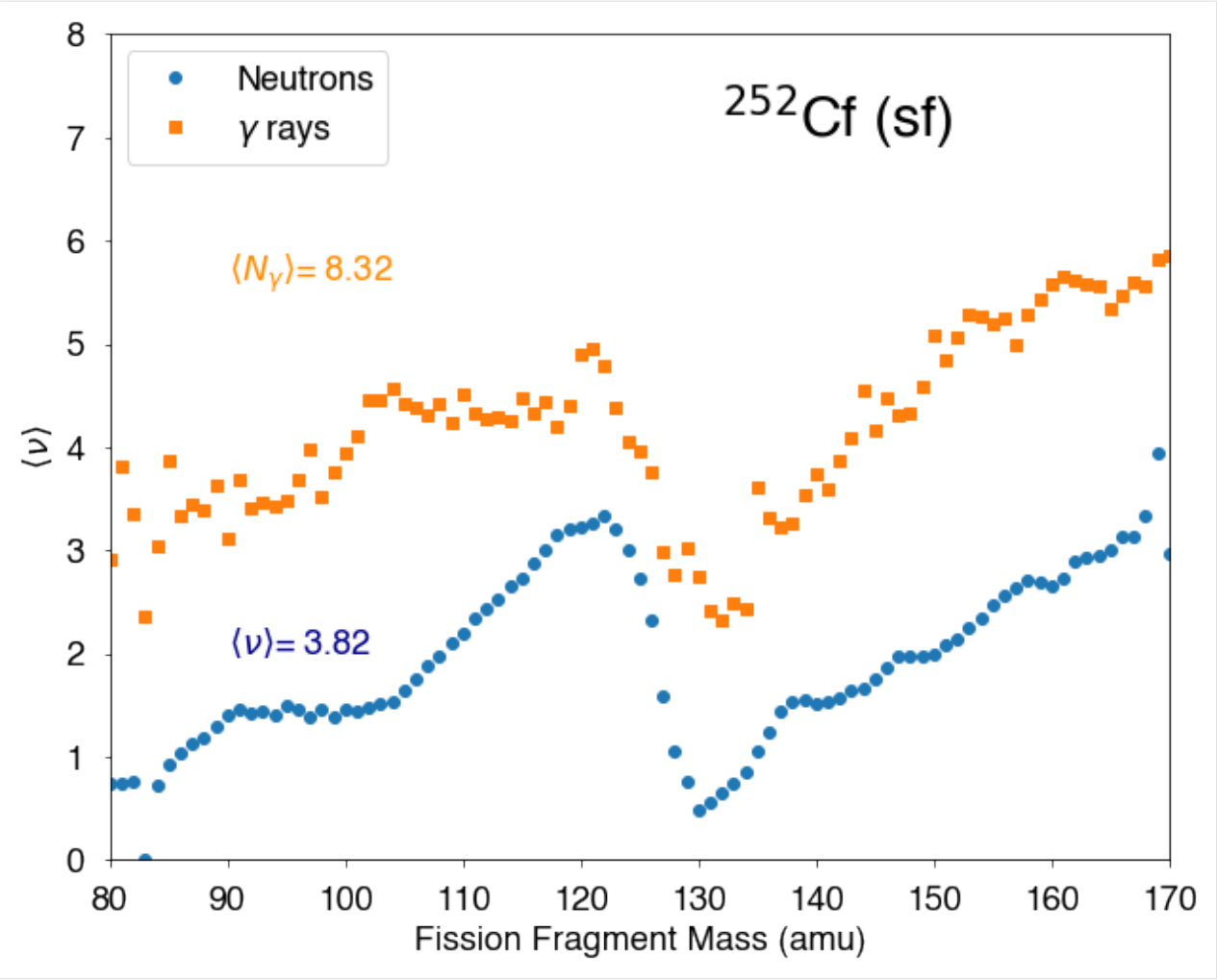
Inferring data as a function of the fragment mass

A useful information is to know how many neutrons are emitted as a function of the mass of the fission fragment. Simply typing:

[4]:
```
A,nu = hist.nubarA()
```

provides the range of A values and the average neutron multiplicity for each A value. It then easy to plot the result:

[7]:
```
fig=figure(figsize(10,8))

ax=plt.subplot(1,1,1)

x1,y1=hist.nubarA()
plt.plot(x1,y1,'o',label="Neutrons")
plt.text(90,2,r"$\langle \nu \rangle$={0:5.2f}".format(2*hist.nubar()),color="darkblue
↪")

x2,y2=hist.nubargA()
plt.plot(x2,y2,'s',label=r"$\gamma$ rays")
plt.text(90,5.6,r"$\langle N_\gamma \rangle$={0:5.2f}".format(2*hist.nubarg()),color=
↪"darkorange")

plt.xlim(80,170)
plt.xlabel("Fission Fragment Mass (amu)")
plt.ylim(0,8)
plt.ylabel(r"$\langle \nu\rangle$")

lg=plt.legend(loc='upper left')
plt.text(132,7,r"$^{252}$Cf (sf)",fontsize=30)
plt.show()
```



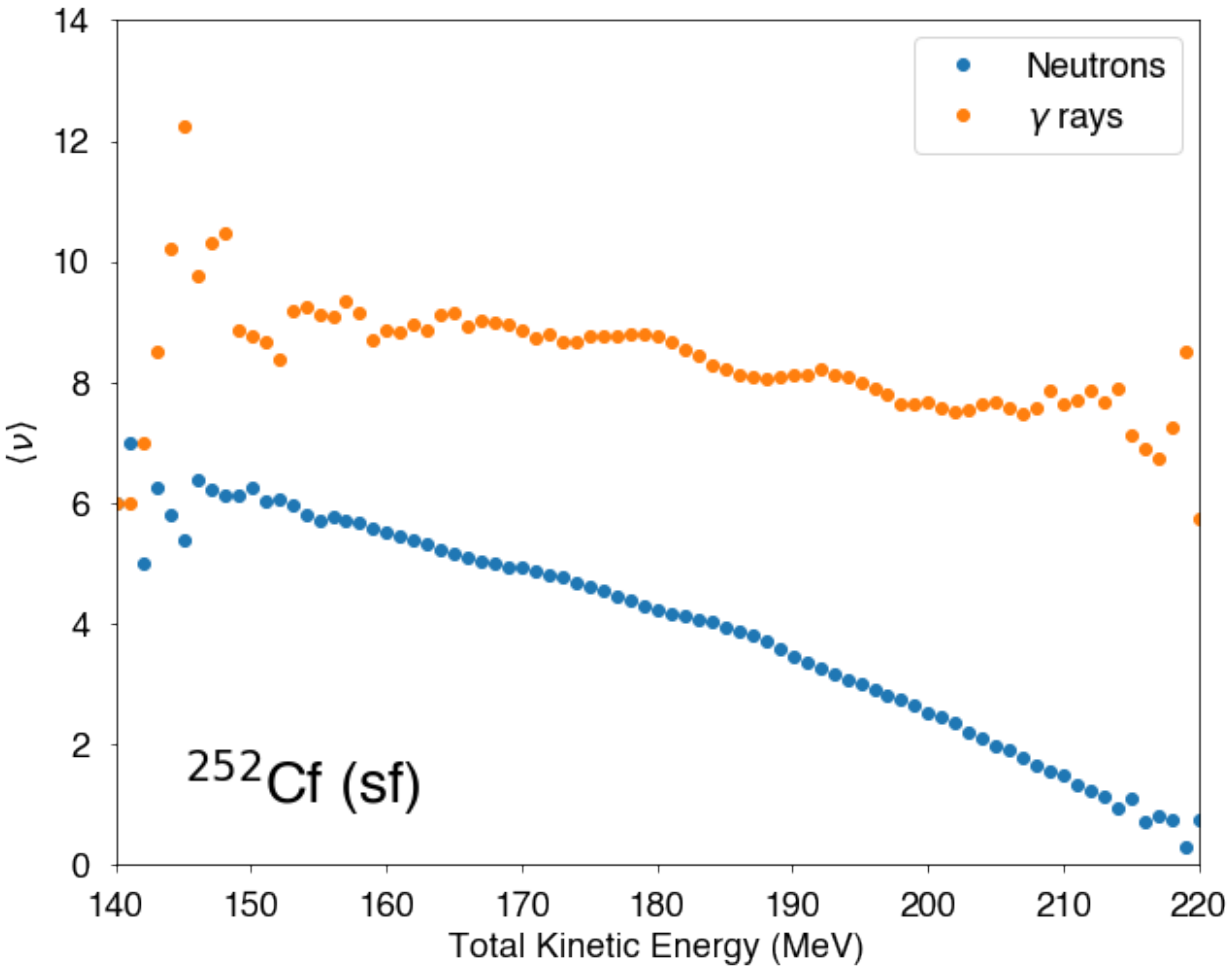
Function of the Total Kinetic Energy (TKE) of the fragments

[14]:
```
fig=figure(figsize(10,8))

ax=plt.subplot(1,1,1)

x1,y1=hist.nubarTKE()
plt.plot(x1,y1,'o',label="Neutrons")

x2,y2=hist.nubargTKE()
plt.plot(x2,y2,'o',label=r"$\gamma$ rays")

plt.xlim(140,220)
plt.xlabel("Total Kinetic Energy (MeV)")
plt.ylim(0,14)
plt.ylabel(r"$\langle \nu\rangle$")

plt.text(145,1,r"$^{252}$Cf (sf)",fontsize=30)

lg=plt.legend()
```

(continues on next page)

(continued from previous page)

```
plt.show()
```



Neutron-neutron angular correlations

For the fission fragment angular distribution with respect to the beam axis/z-axis, there is one option: afterEmission=True/False. afterEmission=True uses these angles after neutron emission and afterEmission=False uses these angles before neutron emission. The default is True.

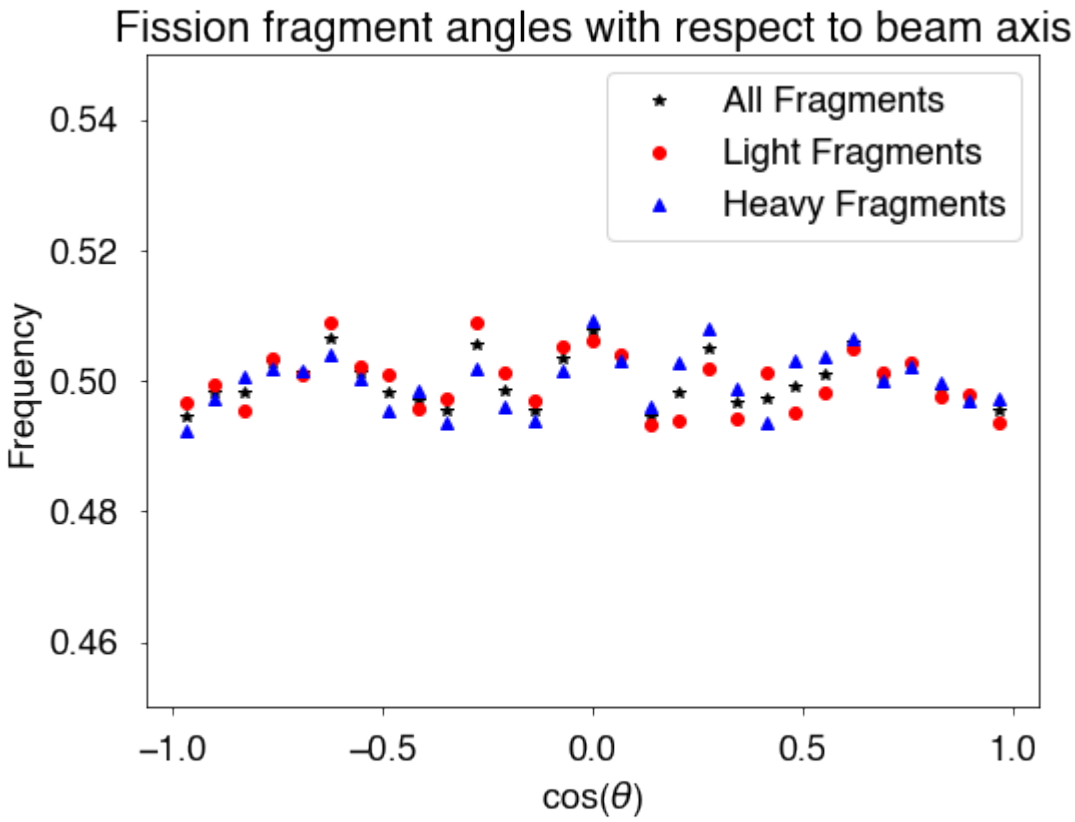
[15]:
```
# calculate cos(theta) between the fragments and z-axis/beam axis
FFangles = hist.FFangles()
bins = np.linspace(-1,1,30)
h,b = np.histogram(FFangles,bins=bins,density=True)
# only light fragments
hLight,b = np.histogram(FFangles[::2],bins=bins,density=True)
# only heavy fragments
hHeavy,b = np.histogram(FFangles[1::2],bins=bins,density=True)
```

[16]:
```
x = 0.5*(b[:-1]+b[1:])
fig = plt.figure(figsize=(8,6))
```

(continues on next page)

(continued from previous page)

```
plt.plot(x,h,'k*',label='All Fragments')
plt.plot(x,hLight,'ro',label='Light Fragments')
plt.plot(x,hHeavy,'b^',label='Heavy Fragments')
plt.xlabel(r'cos($\theta$)')
plt.ylabel('Frequency')
plt.ylim(0.45,0.55)
plt.title('Fission fragment angles with respect to beam axis')
plt.legend()
plt.show()
```



There are several options when calculating the angles of the neutrons with respect to the beam axis/z-axis. The first is including a neutron threshold energy with keyword, Eth (given in MeV). We can also calculate these angles in the lab frame (lab=True, default) or in the center of mass frame of the compound system (lab=False). Finally, we can include pre-fission neutrons (includePrefission=True, default) or not include them (includePreFission=False). However, the pre-fission neutrons can only be include in the lab frame.

[18]:
```
# calculate the angles between the neutrons and the z-axis/beam axis
nAllLab,nLLab,nHLab = hist.nangles(lab=True) # all neutrons, from the light fragment,␣
↪from the heavy fragment
nAllCM,nLCM,nHCM = hist.nangles(lab=False) # center of mass frame of the compound
```

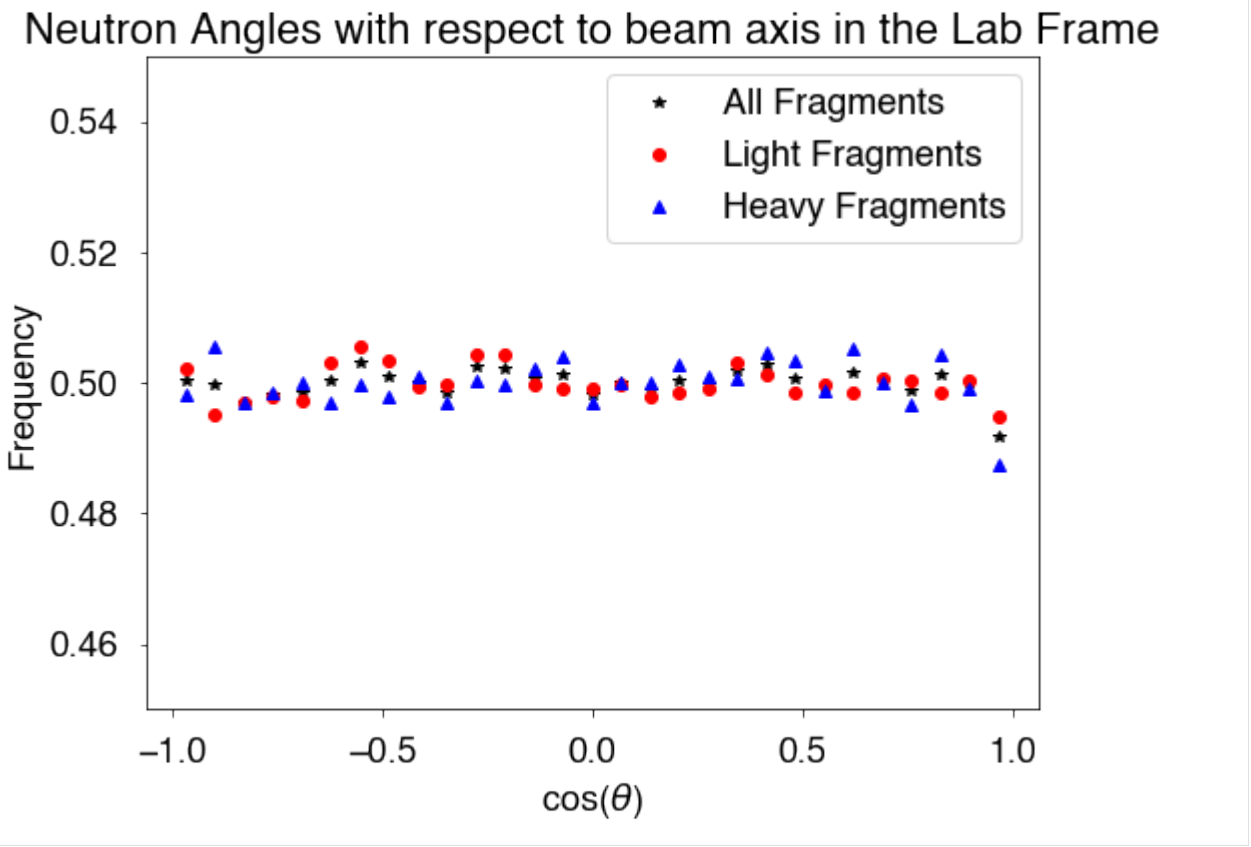
[19]:
```
bins = np.linspace(-1,1,30)
hAllLab,b = np.histogram(nAllLab,bins=bins,density=True)
hLightLab,b = np.histogram(nLLab,bins=bins,density=True)
hHeavyLab,b = np.histogram(nHLab,bins=bins,density=True)
hAllcm,b = np.histogram(nAllCM,bins=bins,density=True)
hLightcm,b = np.histogram(nLCM,bins=bins,density=True)
```

(continues on next page)

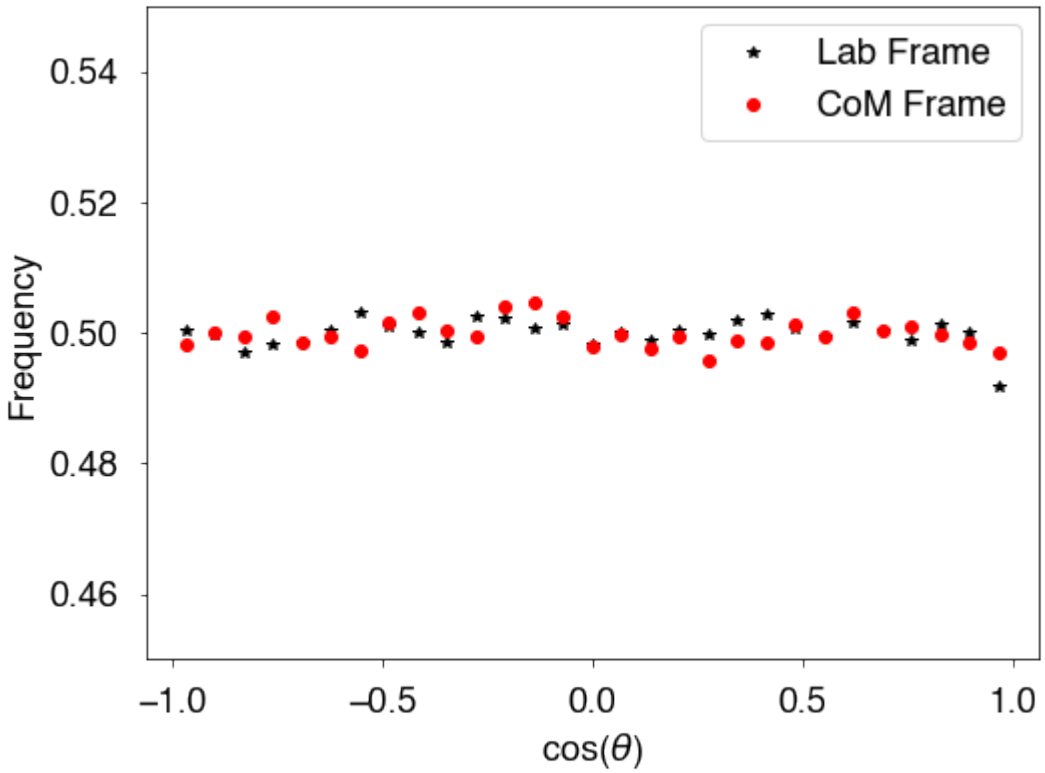
(continued from previous page)

```
hHeavycm,b = np.histogram(nHCM,bins=bins,density=True)
```

[20]:
```
x = 0.5*(b[:-1]+b[1:])
fig = plt.figure(figsize=(8,6))
plt.plot(x,hAllLab,'k*',label='All Fragments')
plt.plot(x,hLightLab,'ro',label='Light Fragments')
plt.plot(x,hHeavyLab,'b^',label='Heavy Fragments')
plt.xlabel(r'cos($\theta$)')
plt.ylabel('Frequency')
plt.ylim(0.45,0.55)
plt.title('Neutron Angles with respect to beam axis in the Lab Frame')
plt.legend()
plt.show()
```



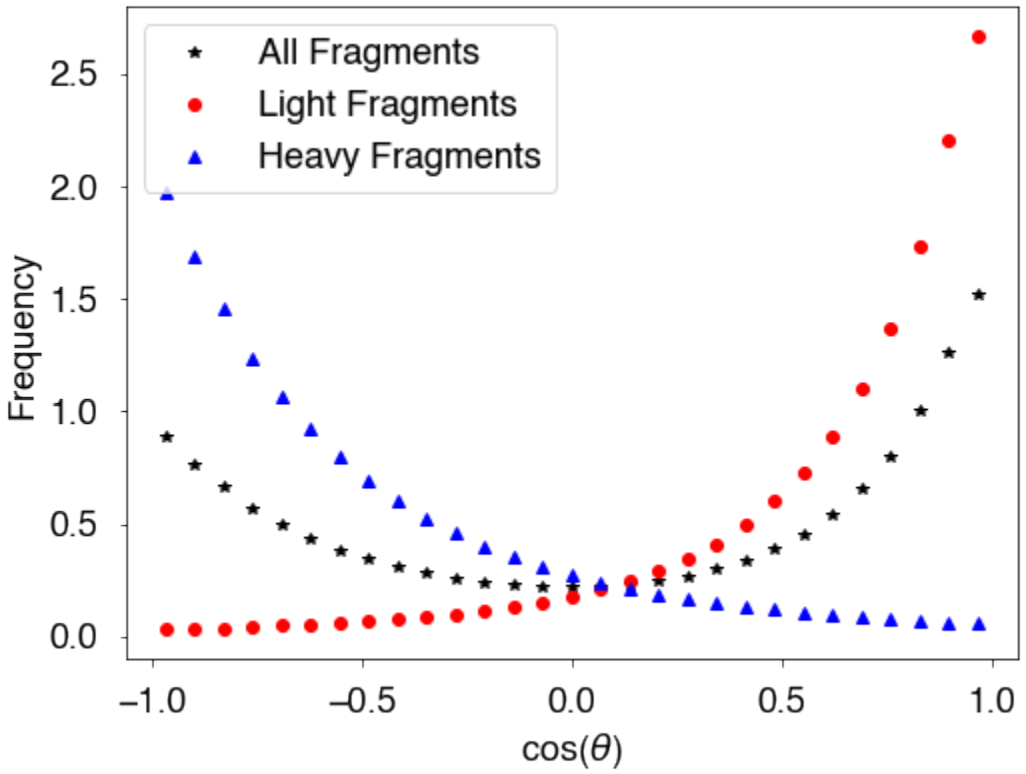
[21]:
```
fig = plt.figure(figsize=(8,6))
plt.plot(x,hAllLab,'k*',label='Lab Frame')
plt.plot(x,hAllcm,'ro',label='CoM Frame')
plt.xlabel(r'cos($\theta$)')
plt.ylabel('Frequency')
plt.ylim(0.45,0.55)
plt.legend()
plt.show()
```



There are again several options that we can use when calculating the angles between all pairs of neutrons (from all framgments) and the ligh fragments, all of which have been seen in the last two examples. These include, Eth (neutron threshold energy), afterEmission (fission fragment angles are post or pre neutron emission), and includePrefission (to include or not include pre-fission neutrons).

[23]:
```
# calculate the angles between the neutrons and the light fragments
nFall,nFLight,nFHeavy = hist.nFangles()
bins = np.linspace(-1,1,30)
hall,b = np.histogram(nFall,bins=bins,density=True)
hlight,b = np.histogram(nFLight,bins=bins,density=True)
hheavy,b = np.histogram(nFHeavy,bins=bins,density=True)
```

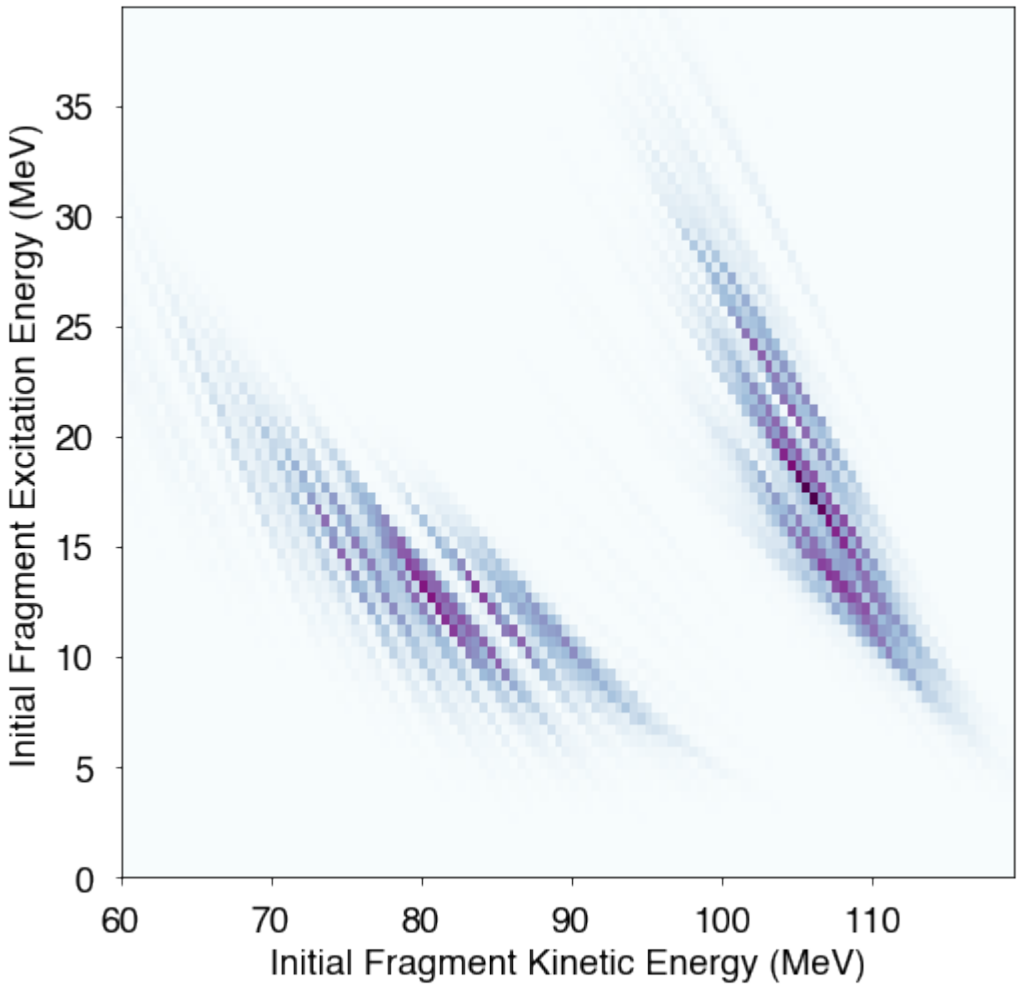
[24]:
```
x = 0.5*(b[:-1]+b[1:])
fig = plt.figure(figsize=(8,6))
plt.plot(x,hall,'k*',label='All Fragments')
plt.plot(x,hlight,'ro',label='Light Fragments')
plt.plot(x,hheavy,'b^',label='Heavy Fragments')
plt.xlabel(r'cos($\theta$)')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



2D correlations

It is also straightforward to create 2D plots of correlations between distributions. For instance, if one is interested in plotting the correlation between the total excitation energy of the fragments and their total kinetic energy, one can simply create a 2D histogram

[25]:
```
# plot the excitation energy against the fragment kinetic energy

KE = hist.getKEpre()
U  = hist.getU()
bx = np.arange(60,120,0.5)
by = np.arange(0,40,0.5)

fig = plt.figure(figsize=(8,8))

plt.hist2d(KE,U,bins=(bx,by),cmap=plt.cm.BuPu)
#plt.ylim(0,35)
plt.xlabel("Initial Fragment Kinetic Energy (MeV)")
plt.ylabel("Initial Fragment Excitation Energy (MeV)")

plt.show()
```



[ ]:

1.6.9 Gammas with Timing Information

[1]:
```
### initializations and import libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
%pylab inline

from CGMFtk import histories as fh
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
### rcParams are the default parameters for matplotlib
import matplotlib as mpl

print ("Matplotbib Version: ", mpl.__version__)
```

(continues on next page)

(continued from previous page)

```
mpl.rcParams['font.size'] = 18
mpl.rcParams['font.family'] = 'Helvetica', 'serif'
#mpl.rcParams['font.color'] = 'darkred'
mpl.rcParams['font.weight'] = 'normal'

mpl.rcParams['axes.labelsize'] = 18.
mpl.rcParams['xtick.labelsize'] = 18.
mpl.rcParams['ytick.labelsize'] = 18.
mpl.rcParams['lines.linewidth'] = 2.

font = {'family' : 'serif',
        'color'  : 'darkred',
        'weight' : 'normal',
        'size'   : 18,
        }

mpl.rcParams['xtick.major.pad']='10'
mpl.rcParams['ytick.major.pad']='10'

mpl.rcParams['image.cmap'] = 'inferno'
```

```
Matplotbib Version:  3.1.3
```

This time, **CGMF** was run with the option `-t -1` to record all of the γ-ray timing information. **CGMFtk** can read these files as well, based on the information that appears in the first line of the file.

[4]:
```
hist = fh.Histories('98252sf_timing.cgmf')
```

```
98252sf_timing.cgmf
```

The timing information can easily be recovered. The prompt γs have a time of 0.

[7]:
```
gammaAges = hist.getGammaAges()
print (gammaAges[:10])
nug = hist.getNug()
print (nug[:10])
```

```
[list([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.89589e-09])
 list([0.0, 0.0, 1.42003e-09, 1.42041e-09])
 list([0.0, 7.42295e-12, 7.34788e-11, 1.94532e-09]) list([0.0, 0.0])
 list([0.0, 0.0, 0.0, 0.0]) list([0.0]) list([0.0])
 list([0.0, 1.17539e-12, 1.54653e-07, 1.54653e-07])
 list([0.0, 0.0, 0.0, 0.0, 0.0]) list([])]
[7 4 4 2 4 1 1 4 5 0]
```

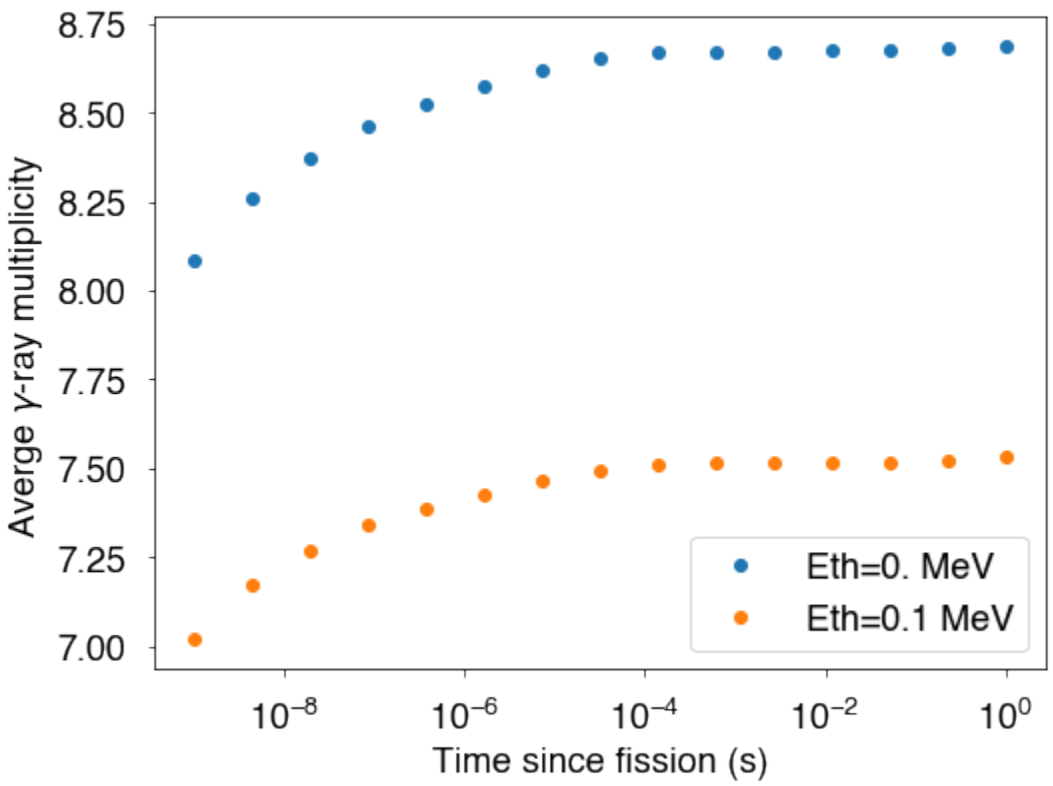
The nubargtot function can also be used to construct the average gamma-ray multiplicity per fission event as a function of time. In the call to nubarg() or nubargtot(), timeWindow=True should be included which uses the default timings provided in the function (otherwise, passing a numpy array or list of times to timeWindow will use those times). Optionally, a minimum gamma-ray energy cut-off can also be included, Eth.

[8]:
```
times,nubargTime = hist.nubarg(timeWindow=True) # include timeWindow as a boolean or␣
↪list of times (in seconds) to activate this feature
fig = plt.figure(figsize=(8,6))
plt.plot(times,nubargTime,'o',label='Eth=0. MeV')
times,nubargTime = hist.nubarg(timeWindow=True,Eth=0.1)
plt.plot(times,nubargTime,'o',label='Eth=0.1 MeV')
plt.xlabel('Time since fission (s)')
plt.ylabel(r'Averge $\gamma$-ray multiplicity')
```

(continues on next page)

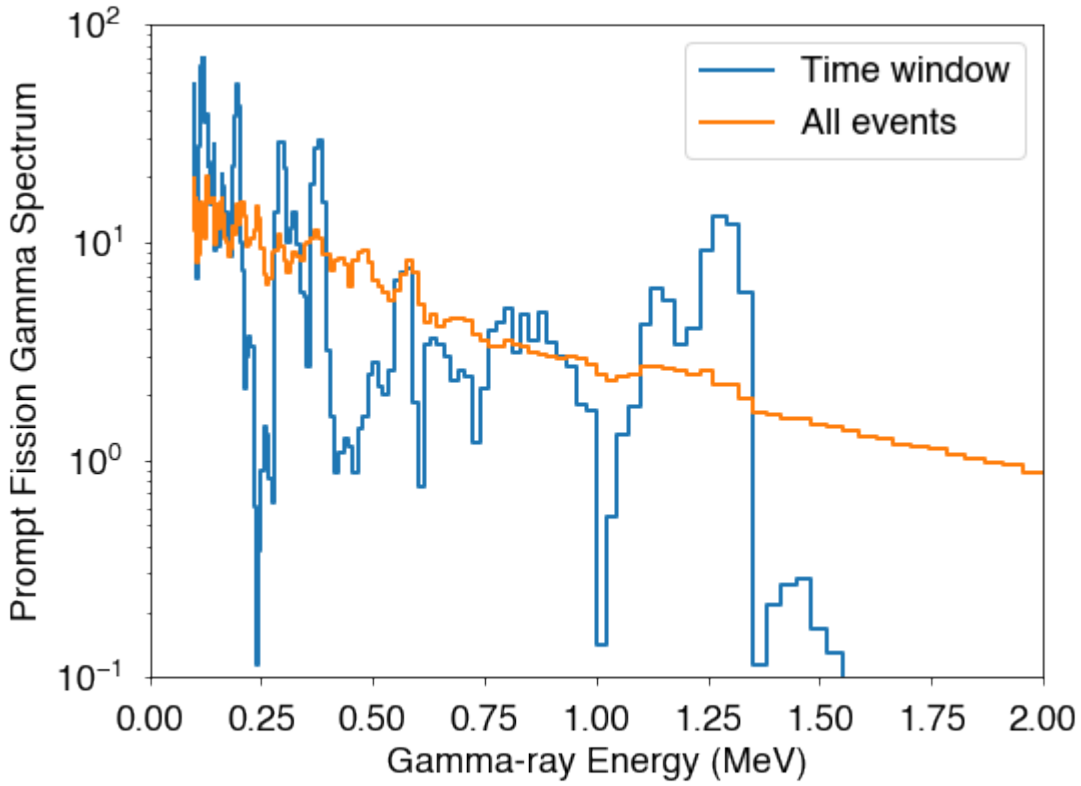
(continued from previous page)

```
plt.xscale('log')
plt.legend()
plt.show()
```



The prompt fission gamma-ray spectrum function, pfgs(), can also be used to calculate this quantity within a certain time window since the fission event. The time window is defined using minTime and maxTime to set the lower and upper boundaries.

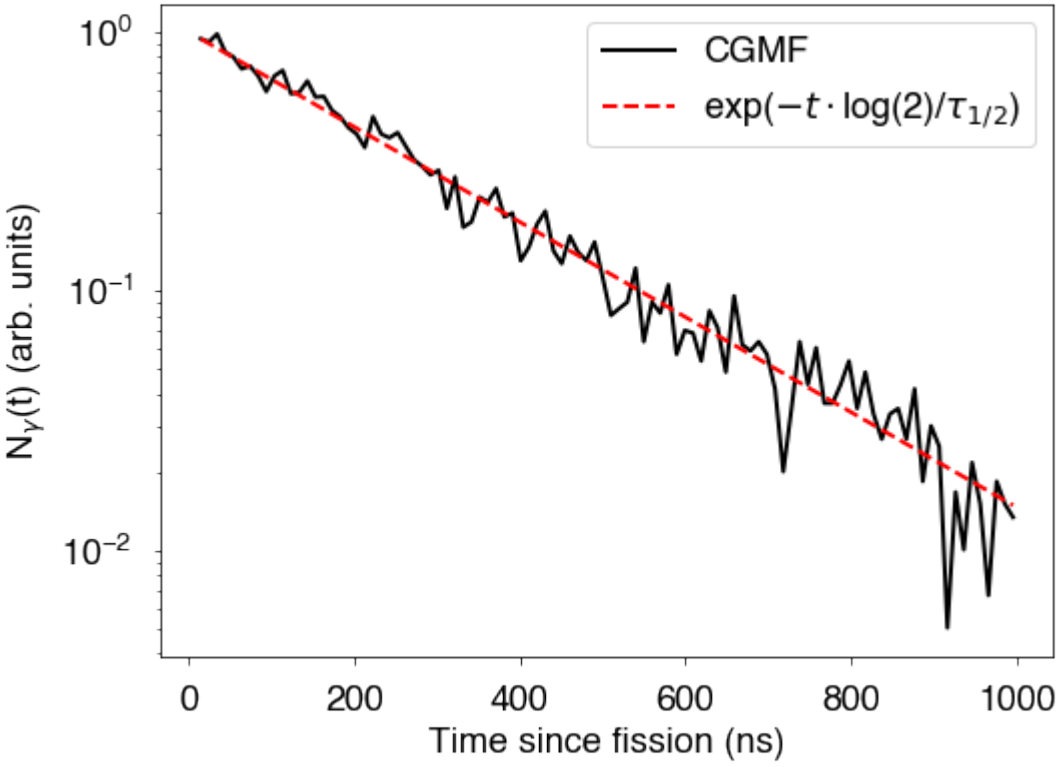
[9]:
```
fig = plt.figure(figsize=(8,6))
bE,pfgsTest = hist.pfgs(minTime=5e-8,maxTime=500e-8)
plt.step(bE,pfgsTest,label='Time window')
bE,pfgsTest = hist.pfgs()
plt.step(bE,pfgsTest,label='All events')
plt.yscale('log')
plt.xlim(0,2)
plt.ylim(0.1,100)
plt.xlabel('Gamma-ray Energy (MeV)')
plt.ylabel('Prompt Fission Gamma Spectrum')
plt.legend()
plt.show()
```



The multiplicity as a function of time can be calculated using the `gammaMultiplicity()` function. The smoothness of the resulting curve will depend on the number of events included in the history file. Here, we only have 500,000 events, which is not quite enough for this type of information.

The fission fragment of interest has to be specified (here A=134, Z=52), and a timing window, in seconds, is given from the minTime to the maxTime (here 10 ns to 1000 ns).

[10]:
```
# calculate the gamma-ray multiplicity as a function of time since fission for a␣
↪specific fission fragment
times,gMultiplicity = hist.gammaMultiplicity(minTime=1e-8,maxTime=1e-6,Afragment=134,
↪Zfragment=52)

# also compare to an exponential decay with the half life of the state
f = np.exp(-times*np.log(2)/1.641e-7) # the half life of 134Te is 164.1 ns
norm = gMultiplicity[0]/f[0]

fig = plt.figure(figsize=(8,6))
plt.plot(times*1e9,gMultiplicity/norm,'k-',label='CGMF')
plt.plot(times*1e9,f,'r--',label=r'exp($-t\cdot$log(2)/$\tau_{1/2}$)')
plt.legend()
plt.yscale('log')
plt.xlabel('Time since fission (ns)')
plt.ylabel(r'N$_\gamma$(t) (arb. units)')
plt.show()
```



Isomeric ratios can also be calculated with **CGMFtk** with the `isomericRatio()` function. Again, the fission fragment needs to be specified, along with the spin of the ground state (`Jgs`) and the spin of the isomeric state (`Jm`). In addition, a threshold time, in seconds, is defined. This time should be significantly shorter than the lifetime of the state, and one should make sure to check various thershold times to ensure the isomeric ratio is independent of the choice of threshold time.

[12]:
```
# calculate the isomeric ratios for specific states in nuclei
# e.g. isomeric ratio for the 1/2- state in 99Nb, ground state is 9/2+, lifetime is␣
↪150 s
r = hist.isomericRatio(thresholdTime=1,A=99,Z=41,Jm=0.5,Jgs=4.5)
print ('99Nb:',round(r,2))
# e.g. isomeric ratio for the 11/2- state in 133Te, ground state is 3/2+, lifetime is␣
↪917.4 s
r = hist.isomericRatio(thresholdTime=1,A=133,Z=52,Jm=5.5,Jgs=1.5)
print ('133Te:',round(r,2))
```

```
99Nb: 0.91
133Te: 0.32
```

[ ]:
```
```

How to add notebook tutorials

To add notebooks to this list, just edit the *examples_notebook.rst* file and link an index entry to a notebook file (.ipynb).

1.7 Publications

- *"Fission Fragment Decay Simulations with the CGMF Code," P.Talou, I.Stetcu, P.Jaffke, M.E.Rising, A.E.Lovell, and T.Kawano, submitted to Comp. Phys. Comm. (Sep. 2020).*
- "Correlations between fission fragment and neutron anisotropies in neutron-induced fission," A.E.Lovell, P.Talou, I.Stetcu and K.J.Kelly, Phys. Rev. C **102**, 024621 (2020).
- "Prompt neutron multiplicity distributions inferred from gamma-ray and fission fragment energy measurements," A.E.Lovell, I.Stetcu, P.Talou, G.Rusev, and M.Jandel, arXiv:1906.05418, Phys. Rev. C **100**, 054610 (2019).
- "Sensitivity of neutron observables to the model input simulations of 252Cf(sf)," J.Randrup, P.Talou, and R.Vogt, Phys. Rev. C **99**, 054619 (2019).
- "Prompt fission product yields in the 238U(n,f) reaction," N.Fotiades, P.Casoli, P.Jaffke, M.Devlin, R.O.Nelson, T.Granier, P.Talou and T.Ethvignot, Phys. Rev. C 99, 024606 (2019).
- "Dependence of the prompt fission gamma-ray spectrum on the entrance channel of compound nucleus: Spontaneous vs. neutron-induced fission," A.Chyzh, P.Jaffke, C.Y.Wu, R.A.Henderson, P.Talou, I.Stetcu, J.Henderson, M.Q.Buckner, S.A.Sheets, R.Hughes, B.Wang, J.L.Ullmann, S.Mosby, T.A.Bredeweg, A.C.Hayes-Sterbenz, J.M.O'Donnell, Phys. Lett. B 782, 652 (2018).
- "235U(n,f) Independent Fission Product Yield and Isomeric Ratio Calculated with the Statistical Hauser-Feshbach Theory," S.Okumura, T.Kawano, P.Jaffke, P.Talou and S.Chiba, J. Nucl. Sci. Tech., DOI: 10.1080/00223131.2018.1467288 (2018).
- "Correlated fission data measurements with DANCE and NEUANCE," M. Jandel, B. Baramsai, T.A. Bredeweg, A. Couture, A. Favalli, A.C. Hayes, K.D. Ianakiev, M.L. Iliev, T. Kawano, S. Mosby, G. Rusev, I. Stetcu, P. Talou, J.L. Ullmann, D.J. Vieira, C.L. Walker, and J.B. Wilhelmy, Nucl. Inst. Meth. in Phys. Research, A **882**, 105 (2018).
- "Hauser-Feshbach fission fragment de-excitation with calculated macroscopic-microscopic mass yields," P. Jaffke, P. Möller, P. Talou, and A.J. Sierk, Phys. Rev. C 97, 034608 (2018) [also on the arXiv: arXiv:1712.05511].
- "Measured and simulated 252Cf(sf) prompt neutron-photon competition," M.J. Marcath, R.C. Haight, M. Devlin, P. Talou, I. Stetcu, R. Vogt, J. Randrup, P.F. Schuster, S.D. Clarke, and S.A. Pozzi, Phys. Rev. C 97, 044622 (2018).
- "Correlated Prompt Fission Data in Transport Simulations," P. Talou, R. Vogt, J. Randrup, M.E. Rising, S.A. Pozzi, J. Verbeke, M.T. Andrews, S.D. Clarke, P. Jaffke, M. Jandel, T. Kawano, M.J. Marcath, K. Meierbachtol, L. Nakae, G. Rusev, A. Sood, I. Stetcu, and C. Walker, Eur. Phys. J. A **54**, 9 (2018), LA-UR-17-28181 (rev.2), arXiv:1710.00107v3.
- "Late Time Emission of Prompt Fission Gamma Rays," P. Talou, T. Kawano, I. Stetcu, J. P. Lestone, E. McKigney, and M. B. Chadwick, Phys. Rev. C **94**, 064613 (2016), LA-UR-16-24045, arXiv:1607.00337.
- "Properties of prompt-fission gamma rays," I. Stetcu, P. Talou, T. Kawano, and M. Jandel, Phys. Rev. C **90**, 024617 (2014).
- "Monte Carlo Hauser-Feshbach Predictions of Prompt Fission Gamma Rays - Application to nth+235U, nth+239Pu and 252Cf (sf)," B.Becker, P.Talou, T.Kawano, Y.Danon, and I.Stetcu, Phys. Rev. C **87**, 014617 (2013).
- "Statistical and evaporation models for the neutron emission energy spectrum in the center-of-mass system from fission fragments," T.Kawano, P.Talou, I.Stetcu and M.B.Chadwick, Nuclear Physics **A913**, 51 (2013).
- "Isomer Production Ratios and the Angular Momentum Distribution of Fission Fragments," I.Stetcu, P.Talou, T.Kawano, and M.Jandel, Phys. Rev. C **88**, 044603 (2013).

- "Prompt gamma-ray production in neutron-induced fission of 239Pu," J.L. Ullmann, E.M. Bond, T.A. Bredeweg, A. Couture, R.C. Haight, M. Jandel, T. Kawano, H.Y. Lee, J.M. O'Donnell, A.C. Hayes, I. Stetcu, T.N. Taddeucci, P. Talou, D.J. Vieira, J.B. Wilhelmy, J.A. Becker, A. Chyzh, J. Gostic, R. Henderson, E. Kwan, and C.Y. Wu, Phys. Rev. C **87**, 044607 (2013).
- "Monte Carlo Simulation for Particle and Gamma-Ray Emissions in Statistical Hauser-Feshbach Model," T.Kawano, P.Talou, M.B.Chadwick, and T.Watanabe, J. Nucl. Sci. Tech. **47**, No.5, 462 (2010).

1.8 Copyright Notice

1.9 BSD License Text

Note:

Current Code Version: 1.1

Date: Apr 07, 2022
