XCOM-Serpent interface

Zs. Elter

June 14, 2018

Abstract

This note summarizes how gamma cross section database XCOM can be interfaced with burnup calculations in Serpent2. First the textual input choices of XCOM are reviewed, then the nuclide inventories computed by Serpent2 are discussed, finally some python functions intended to facilitate the interface are presented.

1 Motivation

For materials with several compounds (eg. nuclear spent fuel) generating photon cross sections is not trivial. Although, there exists the NIST XCOM online database and its user friendly GUI (`https://physics.nist.gov/PhysRefData/Xcom/html/xcom1.html`), one may find it tedious to type in several compounds for a mixture, especially if this needs to be done frequently.

2 XCOM

XCOM provides three ways to generate cross section tables for given mixtures:

1. the online GUI
2. the downloadable XCOM Fortran code `https://physics.nist.gov/PhysRefData/Xcom/Text/download.html`
3. by downloading the raw tables for each element, a home-made software can be developed to create the cross sections for mixtures.

Unfortunately neither option 1., nor option 2. allows the reading of material compounds from a separate file, which would facilitate the scripting of XCOM, when the cross section tables are needed for a vast amount of material compositions. On the other hand, option 3. seems like an enormous effort, especially when considering that the same tool already exists (ie. XCOM:)). As an alternative solution one may use xraymu.m, a matlab script `http://www.aprendtech.com/blog/Post3/xraymu2.html` which is based on NIST data. Nevertheless, if using Matlab is not desirable (eg. one has a transport code written in C or python, does don't want to be dependent on Matlab), this is not a suitable alternative.

2.1 Scripting XCOM GUI?

Although one could use libraries like pygui to script the web-interface for XCOM, I would not recommend that solution. The reason is simple: the script wouldn't be portable (the exact locations where to click and where to type may change with the OS settings) and doing GUI scripting is necessarily slow (comparable with the timing of manual typing)

2.2 Modifying the XCOM Fortran code?

Probably the best way to overcome the whole issue is to modify the Fortran code of XCOM, and implement a text input reader function. Then one can call the updated XCOM code from an other application.

2.3 Bash pipeline...

Or we can just follow Master Foo's wisdom (`http://www.catb.org/esr/writings/unix-koans/ten-thousand.html`) and understand the Unix-nature, and save some time of coding.

After dowloading and compiling the XCOM code, one can test how the input is expected: we have to type entries one by one in the command line (which is again quite tedious if we have to do this for 50+ elements and their weight fractions). Of course if we know beforehand what we would like to input, we can pipeline a *printf* command. Below is an example command given for a material with 0.1 w% H and 0.9 %O for which we want to determine the cross sections at 3 energies (0.6, 0.8 and 0.9 MeV):

```
printf 'testname\n4\n2\nH\n0.1\nO\n0.9\n1\n3\n1\n3
        \n0.6\n0.8\n0.9\nN\ntest.out\n1\n' | ./XCOM
```

where the input entries (seperated by a newline) are the following

- testname : Name of the mixture
- 4 : indicating that we want a composition
- 2 : number of components (in this case H and O)
- H : symbol of element
- 0.1 : weight fraction
- O
- 0.9
- 1 : accept this
- 3 : indicating that we provide the energy list
- 1 : which will be typed in from keyboard

- 3 : number of energy lines
- 0.6 : energy in MeV
- 0.8
- 0.9
- N : No, don't save this into file
- test.out : name of the output file
- 1 : bye bye

Clearly our only task is now to generate and run such bash commands from an other application, and we can consider the above command as a textual input.

3 Serpent nuclide inventories

In case one makes a depletion calculation in an infinite lattice, Serpent provides the nuclide inventory in "bumat" files, which contain a list of the isotopes in the material (with ZAID format, ie. Z*1000+A, however note that for metastable states this differs, eg. Am-244m is 95344. also the ".15c" tag refers to the temperature) and their atomic concentration.

```
% Material compositions (42.81 MWd/kgU / 10372.19 days)

% Decay isotopes not included

mat  UOXp1r1  7.13213323241860E-02 vol 5.28102E-01
            1001.15c  1.95246583900574E-08
            1002.15c  2.43751343073928E-09
            1003.15c  1.69587561240812E-12
            2003.15c  4.88363435398464E-12
            2004.15c  1.18368281744818E-05
            3006.15c  4.12068582428105E-16
            3007.15c  3.65534683336307E-19
            4009.15c  4.15539213459233E-12
            5010.15c  3.54205392240660E-22
            5011.15c  2.59789915757435E-15
            6012.15c  2.18856164447149E-08
            7014.15c  5.97986764318766E-12
            etc..
```

The inventory is given as atomic densities for each isotope with units $b^{-1}cm^{-1} = \frac{1}{b \cdot cm} = \frac{1}{10^{-24}cm^3} = \frac{10^{24}}{cm^3}$

One can relate these values to actual mass, if one knows the volume of the fuel. For example if the calculated Eu-154 atomic density for a PWR fuel is: $AD = 4.22 \cdot 10^{-7}$ $[\frac{10^{24}}{cm^3}]$, then this atomic density corresponds to 6gr of Eu-154 in one assembly: $154g \cdot \frac{AD \cdot *0.41^2 cm^2 \cdot \pi 400cm \cdot 264}{6 \cdot 10^{23}} = 6.04g$ (we have 264 rods in the assembly, radii of pellets are 0.41cm, axial length is 4m).

NOTE: currently the volume of the fuel is considered independently with BU, this has to be handled in future.

What we need to interface?

We would like to put the "material definition" provided by Serpent into XCOM through the above mentioned one liner. For this we need to shape the material definition a bit:

- read the serpent inventory
- convert the atomic densities for each isotope into weight fractions for each element

Reading serpent

Here is a short function to read the bumat files. Note that we jump through the header of the bumat (if one wants to get the BU/CT information from this header, it may be necessary to process that as well)

```
def readInventory(filename):
    """reads bumat file, and outputs list of ZAID and list of
    atomic concentration in [10^(24)/cm^3]"""
    matfile=open(filename).readlines()
    inventory=[]
    concentration=[]
    for line in matfile[6:]:
        x=line.strip().split()
        inventory.append(int(x[0][:-4]))
        concentration.append(float(x[1]))
    return np.array(inventory),np.array(concentration)
```

Converting into element-wise weight fractions

The following function expects the inventory and related concentrations from the previous function. Then it computes the mass/unit volume for each isotope, changes the ZAID into the symbol of the element, and returns the weight fractions in a dictionary. Note, that during this we have to handle the metastable isotopes

```
def AtConc_to_ZWeightPer(inventory,concentration):
    """the function expects list of isotop ZAIDs and serpent
    isotope concentration list in [10^(24)/cm^3]
    the function works in three steps
    1, calculates the mass per cm3 for each isotope
    2, changes the ZAID into elements and sums the mass of each
    element
    3, returns the normalized mass (ie w%) for each element"""
    NA = 6.022140857E23
    masspervolume = {}
    Zprev=1
    massprev=0
    for iso, conc in zip(inventory,concentration):
        A = iso % 1000
        Z = (iso-A)/1000
        if A >= 300:
            if Z<80:
                A = A-200
            else:
                A = A-100

        massconci = A*((conc*1e24)/NA)  #this gives the mass of
    that isotope in cm3

        if Z == Zprev:
            massprev = massprev+massconci
            Zprev=Z #this line is not much needed:)
        else:
            masspervolume[elementsZtoName[Zprev]]=massprev
            Zprev=Z
            massprev=massconci


    #getting weight%
    summass=sum(masspervolume.values())
    for element in masspervolume:
        masspervolume[element]=masspervolume[element]/summass
    return masspervolume
```

Note that "elementsZtoName" is a dictionary which comes from outside the function. This dictionary stores the Z (key) to Symbol (value) relationship. Such as:

```
elementsZtoName={  109:"Mt",
            108: "Hs",
            107: "Bh",
            106: "Sg",
            105: "Db",
            104: "Rf",
            103: "Lr",
            102: "No",
            ...
            9: "F",
            8: "O",
            7: "N",
            6: "C",
            5: "B",
            1: "H"}
```

Thus "masspervolume[elementsZtoName[Zprev]]", means that the keys of the dictionary *masspervolume* are going to be the element symbols, and the values are going to be the weight fractions.

3.1 Creating the material definition to XCOM

Now we just need to shape the dictionary into XCOM format.

```
def XCOMmaterial(massdic):
    xcomstr=''
    for element in massdic:
        xcomstr=xcomstr+element+'\n'+str(massdic[element])+'\n'
    return xcomstr
```

The main script

The "main" is just calling this funtions in a logical order and then runs XCOM with the created input.

```
from BUmuFunctions import *
import os
import numpy as np

bumatfile = '/home/zsolt/Documents/FEIGN/serpent_files/s134.bumat8'

inv,conc = readInventory(bumatfile)

massconcdic =  AtConc_to_ZWeightPer(inv,conc)

xcomfuelstr = XCOMmaterial(massconcdic)


xcomstr="'spentfuel\n4\n"+str(len(massconcdic))+"\n"+xcomfuelstr+"
    1\n3\n1\n15\n1.000E-01\n1.500E-01\n2.000E-01\n3.000E-01\n4.000E
    -01\n5.000E-01\n6.000E-01\n8.000E-01\n1.000E+00\n1.022E+00\n1
    .250E+00\n1.500E+00\n2.000E+00\n2.044E+00\n3.000E+00\nN\
    ntestpython.out\n1\n'"

path=os.getcwd()
os.chdir(path+'/XCOM/')
os.system('printf '+xcomstr+' | ./XCOMtest')
```

Analysing the results

Ideally one does the previous thing for several bumat files (related to several BU values), and then runs XCOM for each material, reads the attenuation coefficient and organizes it in some convenient data format. Without much explanation I include here a case, when I had 11 bumatfiles (related to 13 BU values) for 4 cases (UOX IE 2.5, 3.5, 4.5 % and MOX)

```
import os
from BUmuFunctions import *

path=os.getcwd()
bumatfiles = {3.5: ['/home/../serpent_files/sBurnup.bumat0',
              '/home/../serpent_files/sBurnup.bumat1',
              ...
              '/home/../serpent_files/sBurnup.bumat19'],
              2.5: ['/home/../serpent_files/sBurnup25.bumat0',
              '/home/../serpent_files/sBurnup25.bumat1',
              ...
              '/home/../serpent_files/sBurnup25.bumat19'],
                4.5: ['/home/../serpent_files/sBurnup45.bumat0',
                '/home/../serpent_files/sBurnup45.bumat1',
                ...
                '/home/../serpent_files/sBurnup45.bumat19'],
                'MOX': ['/home/../serpent_files/sBurnupMOX.bumat0',
                '/home/../serpent_files/sBurnupMOX.bumat1',
                 ...
                '/home/../serpent_files/sBurnupMOX.bumat19']}

energies = [1.000E-01,1.500E-01,2.000E-01,3.000E-01,4.000E-01,5.000
    E-01,6.000E-01,8.000E-01,1.000E+00,1.022E+00,1.250E+00,1.500E
    +00,2.000E+00,2.044E+00,3.000E+00]
BUs = [0,10,20,30,40,50,60,70,80,90,100]
muBU=[]
plt.figure()
plt.hold(True)
muBU={'BU': [], 'IE': []}
for en in energies:
    muBU[en]=[]
for bumat in bumatfiles:

    for bumatfile,BU in zip(bumatfiles[bumat],BUs):

        inv,conc = readInventory(bumatfile)

        massconcdic =  AtConc_to_ZWeightPer(inv,conc)

        xcomfuelstr = XCOMmaterial(massconcdic)

        xcomstr="'spentfuel\n4\n"+str(len(massconcdic))+"\n"+
    xcomfuelstr+"1\n3\n1\n15\n1.000E-01\n1.500E-01\n2.000E-01\n3
    .000E-01\n4.000E-01\n5.000E-01\n6.000E-01\n8.000E-01\n1.000E
    +00\n1.022E+00\n1.250E+00\n1.500E+00\n2.000E+00\n2.044E+00\n3
    .000E+00\nN\nbumu"+str(BU)+".out\n1\n'"


        os.chdir(path+'/XCOM/')
        os.system('printf '+xcomstr+' | ./XCOMtest')

        mu=[]
        mufile=path+'/XCOM/'+'bumu'+str(BU)+'.out'
        for en in energies:
            muBU[en].append(readMu(mufile,6,en))
        muBU['BU'].append(BU)
        muBU['IE'].append(bumat)

import pandas as pd
mudb=pd.DataFrame(muBU)
```

Note, readMu is a simple reader function for the XCOM output, 6 is a parameter stating which column one wants to read. It doesn't read the whole column, but one buy one the elements in the column. The reason for this, because in an other simulation this was convenient for me, and I was lazy to update that for the current task. I do not include the function in this document, but it is available with the other functions.

When one has a convenient DataFrame (for example) to store the data, it is simple to plot, what one wants. For example

```
IE=[2.5,3.5,4.5,'MOX']
colors=['r','g','b','k']
for ie,c in zip(IE,colors):
    subdata=mudb[(mudb['IE']==ie) & (mudb['BU']<=60)]
    plt.plot(subdata['BU'],subdata[0.6]/max(subdata[0.6]),c+'--')
    plt.plot(subdata['BU'],subdata[1.25]/max(subdata[1.25]),c+'-.')
    plt.plot(subdata['BU'],subdata[2.0]/max(subdata[2.0]),c)

plt.savefig('mu_vs_BU-UOX-MOX',dpi=1000)
```

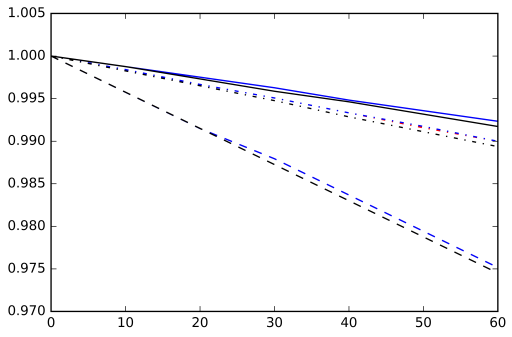

Figure 1: Burnup vs relative attenuation coeff for UOX (blue) and MOX (black) at different gamma energies. Sorry for the lack of labels and legend:)

4 Conclusion

We have a simple tool to quickly evaluate attenuation coefficient of spent fuel at various burnup levels. The tool can be used in other projects to evaluate the gamma spectra of spent fuel, or when using tomography.

One can further investigate the impact of cooling time on the attenuation (should be negligible compared to the impact of BU).

And by including some swelling model, one could include the volume-burnup dependence, and assess what is the total impact on the attenuation coefficient.