Synthetic nTOF Data

Owen Mannion [1]

August 11, 2015

[1]omannion@ur.rochester.edu

0.1 Dependencies

The simulation is was written using Python 3.4 which is open source and can be downloaded at https://www.python.org/downloads/. Python is not notorious for its run time performance and so instead or using the standard python interpreters, the PyPy compiler is used. PyPy is open source and can be downloaded at http://pypy.org/download.html.

With PyPy comes huge gains in computation efficiency but there are also some downsides. The most significant difficultly with PyPy is its lack of compatibility with standard python modules like Numpy and Matplotlib. PyPy is therefore only used to compile/run the simulation which then writes out a .txt file with the results. Then standard python packages can be used to plot all results.

The required packages to run this simulation are:

1. math - Standard with all python environments including PyPy

2. random - Standard with most python environments including PyPy

3. bisect - Standard with most python environments including PyPy

4. numpy - Common scientific package can be downloded at http://www.scipy.org/scipylib/do

5. PypyNumpy - A home made modules that contains all necessary vector operations for pypy. This package is included in this download. This package is modeled after numpy and works very similarly and a user manuel is located at https://github.com/mannion9/Intro-to-Python/tree/master/PypyNumpy

0.2 Structure

The package is structured as shown in Fig. 1. The only two file the user only needs to touch is the Input.py file to make changes to the simulation setup, and run.py to run the simulation. One simply runs the run.py program, which will in turn create the file "AllData.txt" which contains all the relevant results.

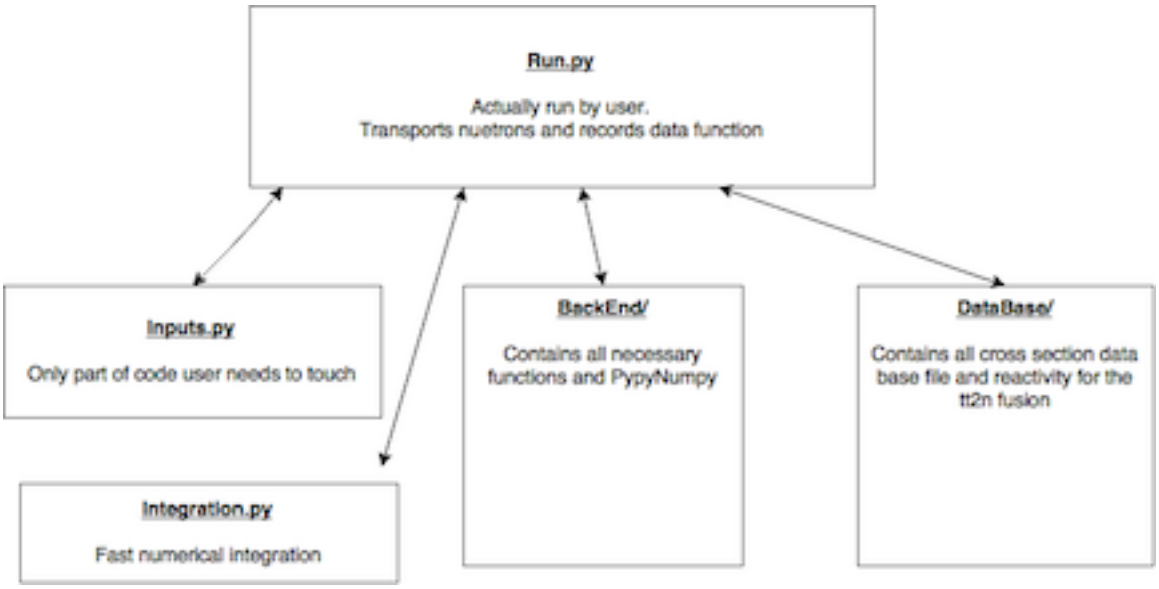

Figure 1: Flow of program.

0.3 User Inputs

The user will use the inputs.py file to create the simulation they desire. This is the only part of the code one must touch.

The current version of the code works for a geometry of an arbitrary number of concentric spheres. Getting the code to run for any given particular geometry may take some time and digging into the code, but once this is done changing other features is quite easy.

The main inputs are:

1. **Number** - Set to the number of neutrons to produce
2. **T_max** - The maximum burn temperature, Units: keV
3. **fusion_list** - A list that allows the user to turn on particular reaction, a 1 indicates the reaction is on and a 0 is off. The indexing is [DT,DD,TT]
4. **Radius** - A list containing the radius of each shell
5. **Density** - A vector containing the density within each shell (note: index of regions must match that of Radius)
6. **A** - A list of lists containing the atomic mass of the atoms within each region (note: list index of regions must match that of Radius)

7. **Ratio** - A vector of lists containing the concentration of each atom within each region (note: list index of regions must match that of Radius, and sublist index must match that of A)

8. **R0** - The radius at which fusions no longer occur

9. **NumberShells** - The number of radius shells to break up the neutron creation into

10. **Burn Width** - The FWHM values of the burn, Units: s

11. **TotalBurnSigma** - The total number of standard deviations of the burn time

12. **TimeStep** - The time steps to break up the neutron creation into, Units: s

13. **Energy_dic_atoms** - A list containing the atom mass of each substance used in the simulation.

14. **DataBase** - The lines of code assigning Cross_Energy, Cross, Energy_dic, Cosines, and Probability are all reading in from text files containing the database information. The index of the lists Cross, Cosines, .ect must match the index of Energy_dic _atoms for the atom for the data being read in.

0.3.1 Database

There are two core components to this simulation that are based on database information. The first is nuclear cross section data, which is used to determine the mean free path. The second is scattering data. Functions have been created to read in both the elastic cross section and the scattering cross section in the NDF/B-VII.1 format. It is key that this format is kept.
An easy way to extract this data in the format needed is to go to http://www.nndc.bnl.gov/sigma/ and select your atom. Then click on the plotting feature for the data you would like. Then on the right click on "view evaluated data" and save the file as a .txt.
Once the file is saved it should be put into the Database folder and the corresponding elastic or scattering folder. Then to read in the data, follow the format of the example data by assigning the n^{th} element of the Cosine, Probability, .ect to the file you would like. Ensure that the index matches that of the n^{th} atom in your Energy_dic_atoms.

0.4 Geometry

The geometry of the simulation is quite simple. The whole "universe" is a perfect sphere. Within this "universe", there are regions each of which are themselves spheres. Each region is specified by a radius, the substance in the region, the density of the substance, and concentration of nuclei that the substance is composed of.
The equation of a surface is of the form $F(\vec{r}) = 0$. For a sphere, this equation is

$$F(x, y, z) = x^2 + y^2 + z^2 - r^2 \tag{1}$$

where r is the radius of the sphere. If a point is inside a sphere, evaluating Eq.(1) would result in a negative number, if a point is outside the sphere Eq.(1) would be positive, and if the point is on the sphere Eq.(1) would be zero.
In our transport algorithm it is important to know the minimum distance a position $\vec{r}$ is to a given surface along a direction $\hat{v}$. To find this distance we will write

$$\vec{r} = \vec{r}_o + d\hat{v} \tag{2}$$

where $\vec{r}$ is a new position created by drawing a ray from $\vec{r}_o$ in the direction of $\hat{v}$ of length d. Now one can decompose Eq.(2) into its corresponding scalar equation. These scalar equations can then be inserted into Eq.(1) reducing Eq.(1) to an equation of one variable, d. For a sphere this becomes

$$F(d) = (x_o + d\hat{v}_x)^2 + (y_o + d\hat{v}_y)^2 + (z_o + d\hat{v}_z)^2 - r^2 \tag{3}$$

which is simply a quadratic equation of d.
As stated above, if $F(\vec{r})$ is equal to zero, then $\vec{r}$ lies on the surface, and so setting Eq.(3) to zero we can solve for the value of d that puts $\vec{r}$ on the surface. This allows us then to find the minimum distance a position $\vec{r}$ is from a surface along a direction $\hat{v}$.

0.5 Neutron Creation

There are three possible neutron production reactions those being DTn, DDn, and TT2n. To decide which reaction is chose a random choice is made based by weighting each reaction by their fusion rate per unit volume per time given by

$$\frac{dN}{dVdt} = \frac{n_i n_j}{1 + \delta_{ij}} < \sigma v > \tag{4}$$

where n_i and n_j are the number densities of the two reactants, δ_{ij} is the Kronecker delta and $< \sigma v >$ is the reactivity of the fusion which is a function of temperature. The reactivity is taken from the Bosch and Hale parametrization.
To determine the number of neutrons produced per unit per time the above equation is first normalized to the total desired number of neutrons, and then numerically integrated to determine the number of neutrons produced per radius per time.
This whole integration is done using python3 and Numpy outside of the main program. This is done to decrease run time because Numpy has extremely fast matrix multiplication.
Once a neutron's reaction and location in space and time is chosen, the energy at which the neutron is born at must be chosen. The birth spectrum used is a modified Gaussian decomposed into Hermite polynomials in the momentum domain. This is done to allow the freedom to set the spectrum's moments to what ever we wish. In particular the mean and variance are set to the formulas derived by Ballabio that relate the moments to the temperature of the ions. The functions are

$$P(x) = \frac{A}{\sqrt{2\pi}\sigma}[H_0(x) + \frac{\gamma_1}{3!}H_3(x) + \frac{\gamma_2}{4!}H_4(x)]e^{\frac{-x^2}{2}} \tag{5}$$

where $x = \frac{p-\mu}{\sigma}$, μ=mean, σ=variance, γ_1 is the skew, γ_2 is the kurtosis, and p is the momentum.
The mean in the energy domain can be shown to be correlated to the temperature by the Ballabio shift given by

$$< E >= E_0 + \Delta E \tag{6}$$

where the zero temperature energy E_0 is

$$E_0 = \frac{(M_3 + Q/2)Q}{M_1 + M_2} \tag{7}$$

and where the Ballabio shift $< \Delta E >$ is

$$\Delta E = \alpha_0 T_{ion}^{2/3}/(1 + \alpha_1 T_{ion}^{\alpha_2}) + \alpha_3 T_{ion} \tag{8}$$

where α are fit parameters given in the Ballabio paper. The variance can be shown to be correlated to the temperature and the mean momentum by

$$\sigma = \frac{2M_n(\sqrt{\mu_o^2 + M_n^2} - M_n)(\mu_o^2 + Mn^2)}{(\mu_o^2(M_n + M_3))}T_{ion} \tag{9}$$

where M_n is the mass of the neutron in MeV, M_3 is the mass of the other outgoing particle, μ_o is the zero temperature birth momentum,

M_{out} is the mass of other outgoing product in the fusion (either α or neutron) and T_{ion} is the temperature of the plasma in MeV.
With these values now defined, the momentum in which a neutron is born at was sampled from the PDF, and then converted into a neutron energy. Note: The skew and kurotsis was assumed to be zero.

0.6 Transporting

A neutrons mean free path is defined as:

$$\lambda = \frac{1}{(N\sigma)} \tag{10}$$

where σ is the total nuclear cross section for the substance the neutron is within, and N is the number density given by

$$N = \frac{N_A \rho}{A} \tag{11}$$

where N_A is Avagadro's number, ρ is the density of the substance, and A is the atomic mass of the atoms within these substance.
At each step the neutron is transport either a mean free path, or to a geometric boundary, whichever is smaller. The mean free path is evaluated using nuclear cross section data inputted by the user, along with the density and atomic mass of the atoms. The distance to a geometric boundary is determined by using the methods described above in the Geometry section, that being finding the roots of Eq. (3) with a known position and velocity direction.
If the neutron is chosen to move a mean free path, then it is transported a mean free path and scatters.
If the neutron is chosen to move to a geometric boundary G, it is transported in a series of steps to that geometric boundary. The size of these steps is a random fraction of G. At each step the neutron has a probability of scattering given by Beer's law

$$P(scatter) = \frac{G}{\lambda} exp(\frac{-x}{\lambda}) \tag{12}$$

where x is the fraction of G that has been traveled to that point.
If a neutron is scattered the energy loss and scattering angle are determined. These then define a new neurton energy and a new velocity direction. The new energy is a function of the scattering angle given by

$$E' = \frac{E}{2}((1+\alpha) + (1-\alpha)cos(\theta_{cm})) \tag{13}$$

where E' is the new energy, E is the incindent energy, θ_{cm} is the scattering angle in the center of mass frame and where α is

$$\alpha = (\frac{A-1}{A+1})^2 \tag{14}$$

where A is the atomic mass of that nuclie that the neutron collides with. The new direction is determined by adding previous steps angle and the scattering angle.
The scattering angle is sampled from ENDF VII databases, which containt a PDF for the scattering angle in the ceneter of mass frame.

0.7 Algorithm

The function is simple in its core function. Every neutron is transported from its birth location to the edge of the "universe" This algo-

Define user geometry
Create Data Base
Birth neutrons (done in python3)
for *every neutron* **do**
  while *neutron is inside the "universe"* **do**
    calculate λ = neutron mean free path
    calculate G = distance to boundary in direction of momentum;
    if $\lambda < G$ **then**
      transport neutron λ;
      scatter;
    else
      η = random number $\in$ [0,1) ;
      x = η*G ;
      transport x ;
      scatter neutron with probability given by Beer's law
    end
  end
end
Write out to AllData.txt

rithm is linear but has a nonlinear element. For example if the mean free path λ is very small, the loop could repeat many times.

0.8 Fuctions

Since Numpy is not compatible with the PyPy compiler, much of the tools Numpy provides had to be hard coded into the simulation. This

included all vector operations and numerical integration. A brief description of each function in the simulation is given below