Preliminary Analysis of Delta Tracking

Hunter Belanger

1 Description of System

The system which I have chosen to build this toy model around is a slab reactor, which is infinite in two dimensions, and finite along the x axis, with vacuum at either end of the reactor. The reactor itself is composed of a core which is a multiplying medium, and a reflector at either side which is non-multiplying. A depiction of this system along the x axis is presented below in Figure 1.

The two group approximation is used in these programs, with the assumption of isotropic scattering for the moment. the cross sections for each material at each energy are provided in tables bellow (with scattering matrices).

1.1 Baseline Solution

Two solutions for the eigenvalue of this system were obtained using MCNP and OpenMC. MCNP was also used to calculate the scalar flux for each neutron group. These calculations were done with 1000000 histories, at 1015 generations (15 of which were ignored).

The criticity from MCNP in Table 3 shall be considered the true eigenvalue for this system for the purposes of this work. The normalized scalar flux from MCNP (with error bars) is presented in Figure 2. The error of the scalar flux is plotted for reference in Figure 3.

1.2 Calculation Methods

Here are outlined the methods used to make the calculations of the flux, entropy, and FOM. The following codes all use the same methods to calculate these values to minimize differences between results that are not directly related to the tracking method (ray-tracing, delta-tracking, etc.).

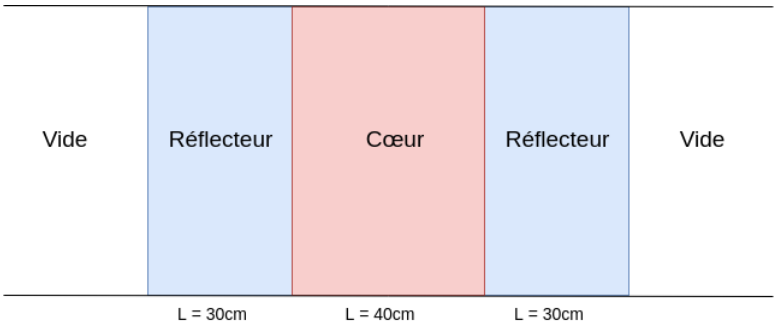

Figure 1: Reflected slab reactor, finite in x axis, infinite in y and z.

Cross Section	Reflector	Core
Σ_{t_1}	0.295	0.276
Σ_{t_2}	2.1	1.063
Σ_{a_1}	0.0004	0.012
Σ_{a_2}	0.02	0.121
Σ_{f_1}	0.0	0.00339
Σ_{f_2}	0.0	0.074
ν_1	0.0	2.5
ν_2	0.0	2.5

Table 1: Cross sections for the system (in units of cm^{-1}), except scattering matrices.

	to 1	to 2
1	0.2456	0.049
2	0.0	2.08

(a) Reflector

	to 1	to 2
1	0.25	0.014
2	0.0	0.942

(b) Core

Table 2: Scattering matrices, $\Sigma_{j \to i}$, for the system (in units of cm^{-1}).

Platform	k_{eff}
MCNP	1.00914 ± 0.00002
OpenMC	1.00915 ± 0.00002

Table 3: Criticality of the system.

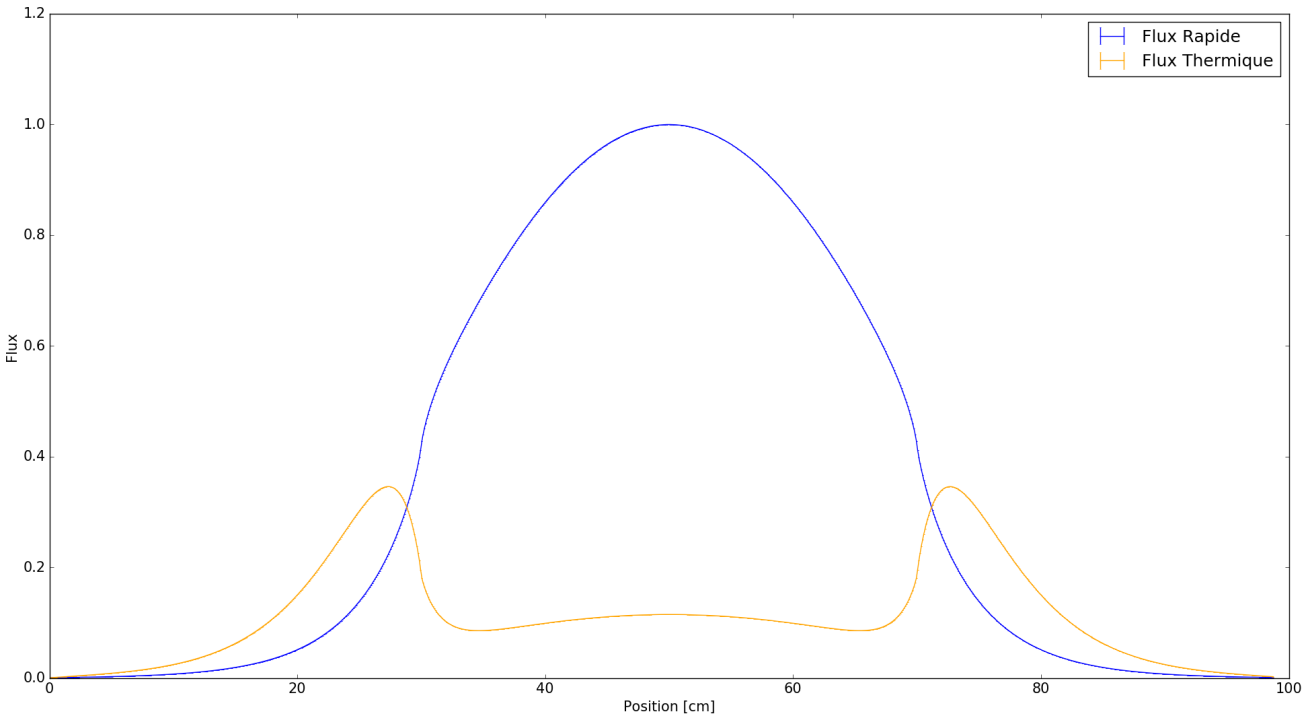

Figure 2: MCNP scalar flux with error bars.

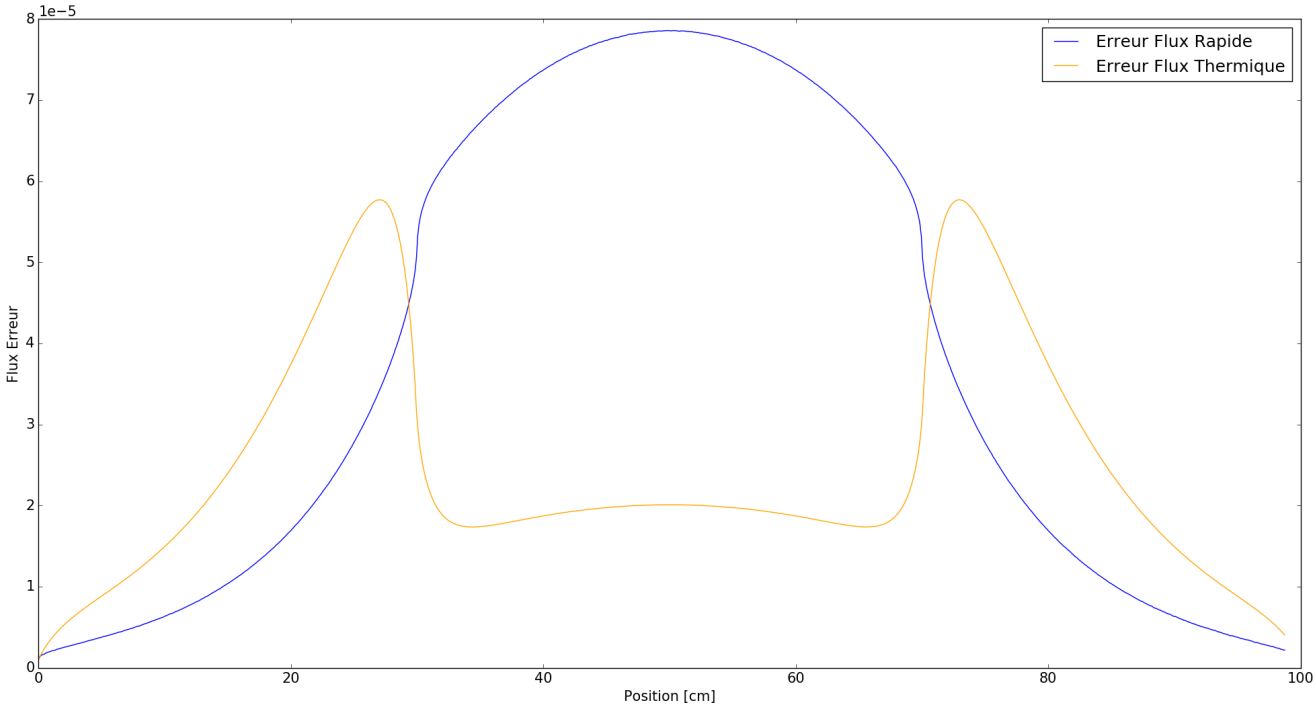

Figure 3: MCNP scalar flux error.

1.2.1 Scalar Flux

To calculate the scalar flux, the core was divided into 1000 bins along its x axis, each with a thickness of $0.1cm$. Each time a neutron underwent a collision, the flux tally in that bin was increased with:

$$F_{E,b} \quad += \quad \frac{w}{\Sigma_t(x)} \tag{1}$$

where $F_{E,b}$ is the flux tally in bin b for energy group E. Bin b corresponds to the location of the collision (x), and w is the weight of the neutron. It is not necessary to normalize the box by size as all boxes have the same size, and only the normalized scalar flux is being considered.

1.2.2 Entropy

In a similar manner to the flux, the entropy was calculated for each generation, with the core region being divided into 200 bins, with a thickness of $0.2cm$. The number of neutrons in box b at the beginning of the generation is N_b, and the value of the entropy for the generation is calculated as

$$H = \sum_b N_b \log_2(N_b) \tag{2}$$

where it is assumed all boxes have at least 1 neutron. This is a valid assumption for these cases with large bins and a large number of histories per generations.

1.2.3 Figure of Merit

The Figure of Merit (FOM) is calculated with the following formula:

$$FOM = \frac{1}{T\left(\frac{\sigma}{k_{avg}}\right)^2} \tag{3}$$

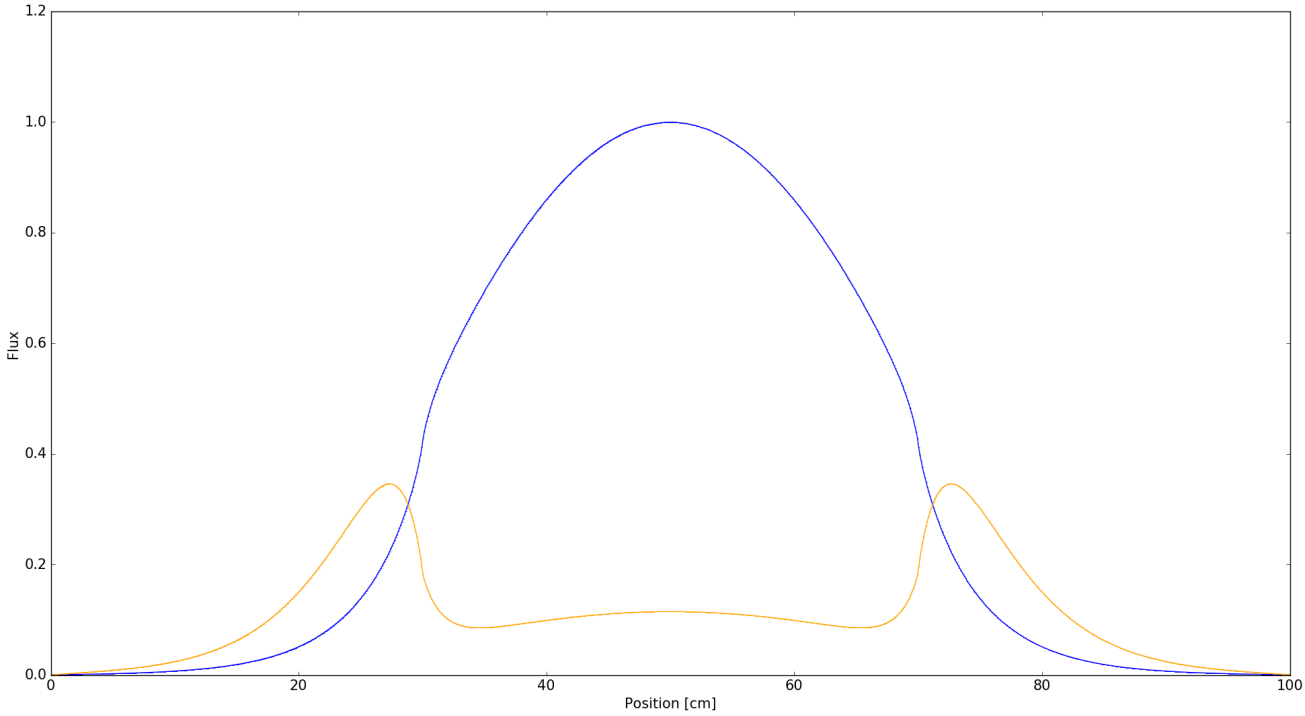

Figure 4: Ray-tracing flux with error bars.

where T is the time in seconds since the first generation which was not ignored began, and σ is the standard deviation of the current average for the criticality of the system, k_{avg}.

1.3 Russian Roulette

As the neutrons are weighted, Russian roulette is played to kill particles who become "unimportant" due to their low weight. This is done with a cutoff weight of 0.25 and a survival weight of 1.0.

2 Ray Tracing Program

The ray-tracing model is found in the "ray_trace" directory. This program was written first, and uses what I believe to be a standard ray-tracing method. The flight distance is calculated with

$$d = -\frac{\ln(\xi)}{\Sigma_T}. \tag{4}$$

The distance to the nearest surface is then calculated based on the particles flight direction, and a comparison is made: If the distance d is shorter than the distance to the closest surface (d_{surf}) then the particle is moved by a distance d and of course stays within the same cell and material. Should the converse be the case, the particle is moved a distance $d_{surf} + \epsilon$ so that it is just on the other side of the surface, and is in the new material. A new travel distance is then calculated as before with the new cross section.

This program came up with an eigenvalue of $k_{eff} = 1.00908 \pm 0.00002$. The plot of the flux and flux error and provided in Figures 4 and 5 respectively.The full simulation output is found in the ray_trace directory, in the text file terminal_output.

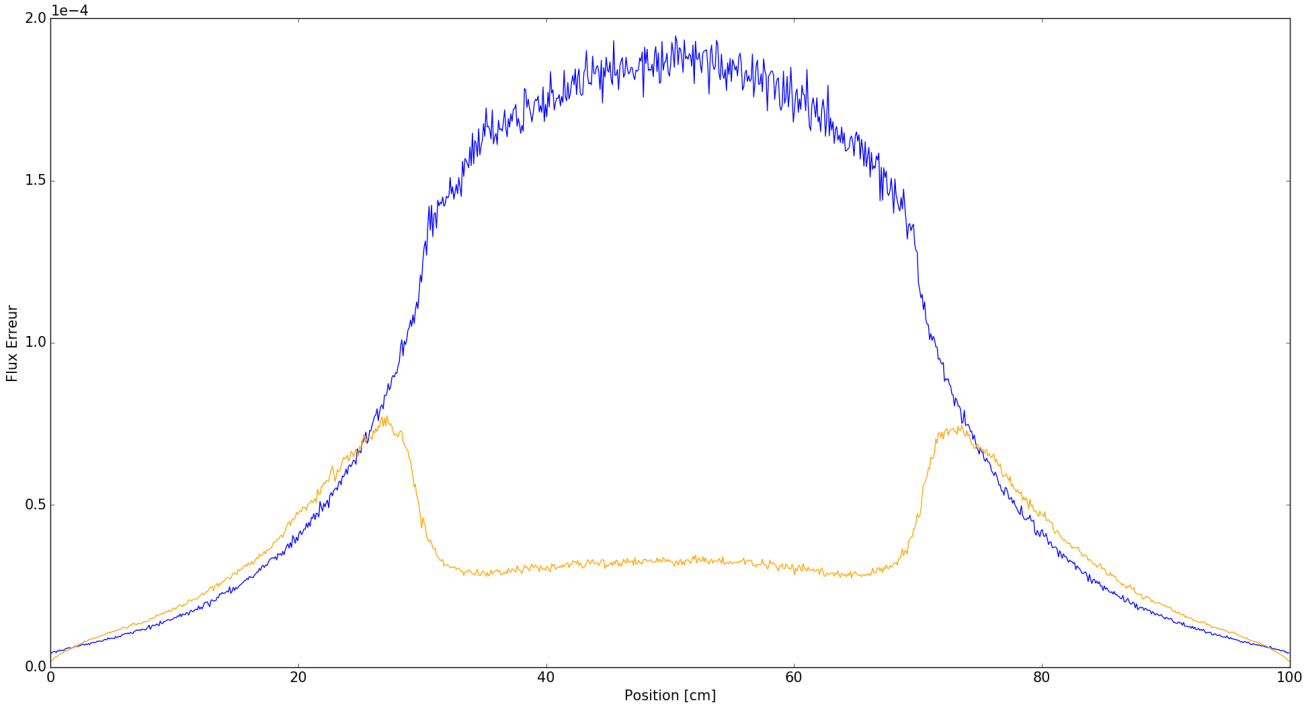

Figure 5: Ray-tracing flux error.

3 Delta Tracking Program

Delta tracking works in a slightly different way than ray-tracing. Instead of making a check to see if the distance to the surface of a cell is shorter than the sampled flight distance, a flight distance which is valid for the entire system is calculated, replacing the material cross section Σ_T with the majorant cross section for the system Σ_M. The particle is then moved that distance, and its current location and material are sampled. Using the actual cross section at the particle location $\Sigma_T(x)$, if

$$\xi < \frac{\Sigma_T(x)}{\Sigma_M}, \quad \xi \in [0, 1) \tag{5}$$

the collision is accepted as being real. Otherwise a virtual collision has occurred. Should this be the case, the particle may then be "kicked" forward an additional distance by calculating a new distance as before. The second option, labeled no_kicking in the delta_tracking directory, simply moves the particle back to it's original location, and samples a new flight distance.

3.1 Particle Escape

Basic delta-tracking as described in papers has one minor oversight, and that is how to deal with particles escaping to a vacuum region. With ray-tracing, it's quite simple: the particle is moved just barley into the vacuum region when it crosses the cell boundary, and it finds that $\Sigma_T = 0$, causing it to have an infinite flight distance and escapes.

Delta tracking does not keep track of when a particle crosses a surface boundary. As such, a particle may cross into vacuum, and a virtual collision check will occur. At this point, the probability of a virtual collision is now exactly zero due to Equation 5. With no change in particle energy or direction, an infinite loop occurs where a collision may never be accepted.

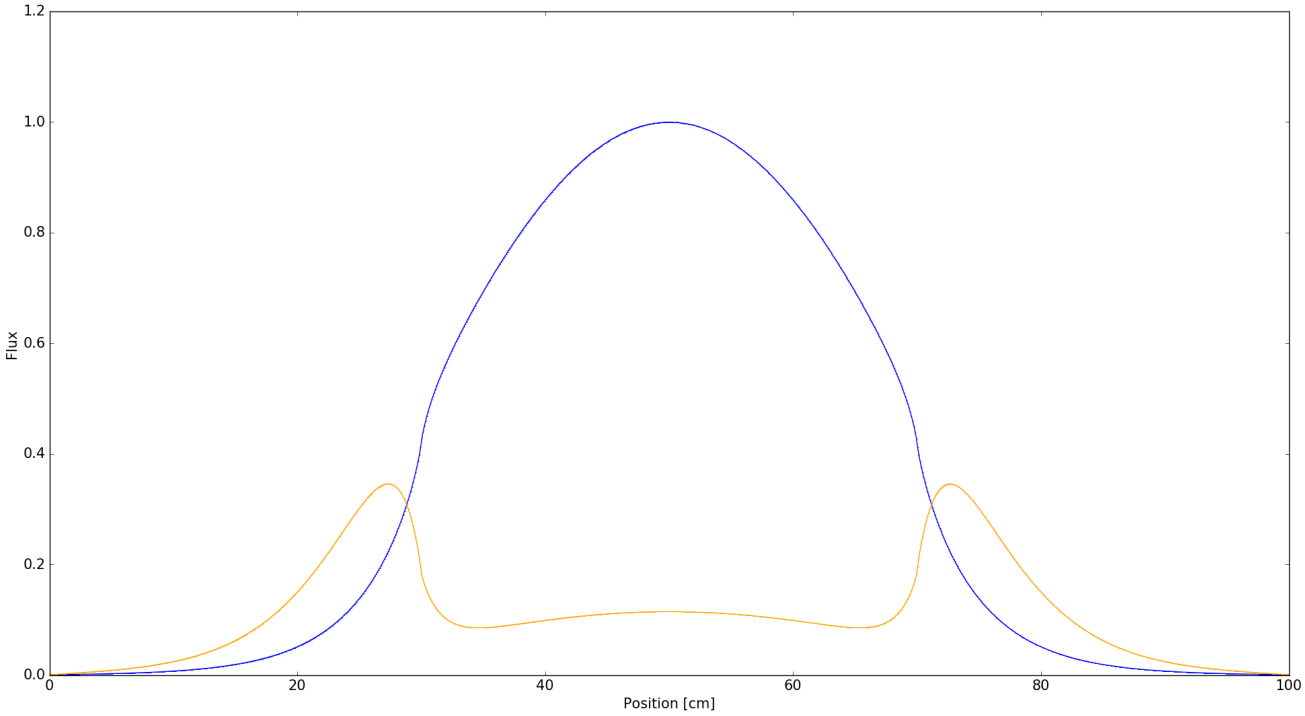

Figure 6: Delta-tracking scalar flux with error bars.

To get around this dilemma, I wrote the program so that if a particle does find itself in a vacuum, it is considered dead and no check for a virtual or real collision is made. This should be a valid assumption to make as it is possible and valid to switch from delta-tracking to ray-tracing for different flights.

3.2 Results

As it turns out, the "kicking" method and "no_kicking" method of delta tracking are equivalent models, due to the exponential functions being memoryless. As such, only the eigenvalue and flux plots for the kicking version of the model are shown. The determined eigenvalue was $k_{eff} = 1/00914 \pm 0.00002$. The flux and flux error are shown in Figures 6 and 7.

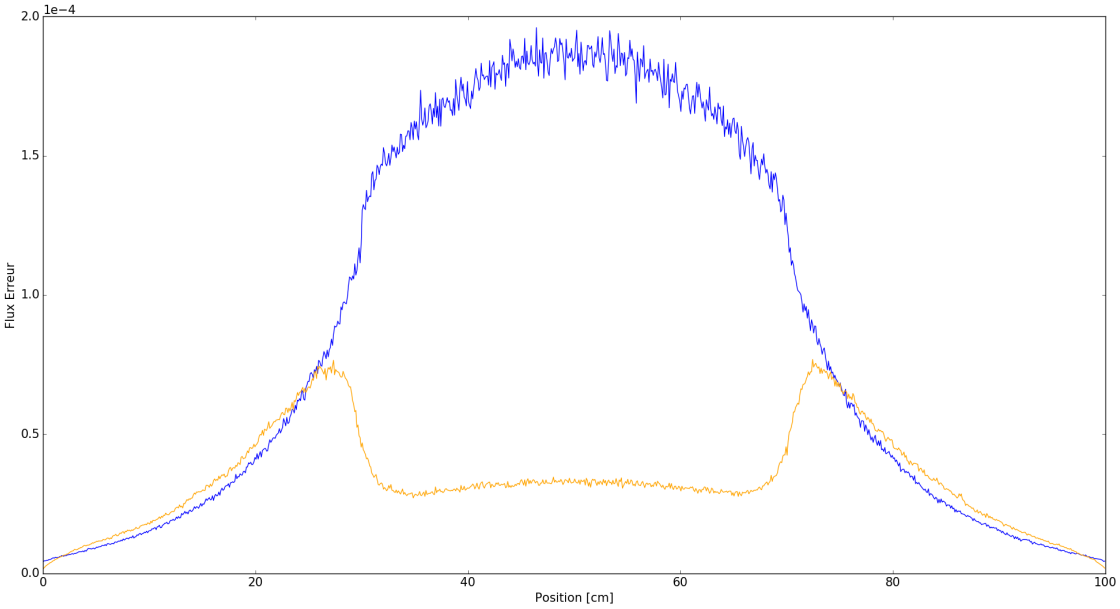

Figure 7: Delta-tracking scalar flux error.