Haotian Zeng, Nanyu Zhao

May 15, 2019

NE-155 Final Project Report

Introduction:

Our goal is to make a simple Monte Carlo model of the neutron transport problem in 2D geometry. The 2D neutron transport equation is a model of neutrons at specific energies and angles within a fixed area. The neutron transport problem has three production terms and two loss terms. Production occurs from scattering, from fission, and from a fixed source, while loss occurs from collisions and leakage from the area.

The equation itself assumes particles are point objects, ignoring rotation and momentum. It assumes particles travel in straight lines and collisions between particles are negligible, making it linearly modellable. It assumes material properties are isotropic and material composition does not change over time. It does not deal with uncertainties.

Additionally, in our code we assume the neutrons are monoenergetic and all scattering is elastic. This eliminates the need to model neutron energy, and because we are finding a scalar flux, we do not need to keep track of the angle either.

In our code, we specify an initial number of neutrons to generate, and the initial source boundaries they are uniformly distributed within. We also specify constant rectangular bounds, outside which neutrons are considered leaked, as well as cross sections for absorption and scattering.

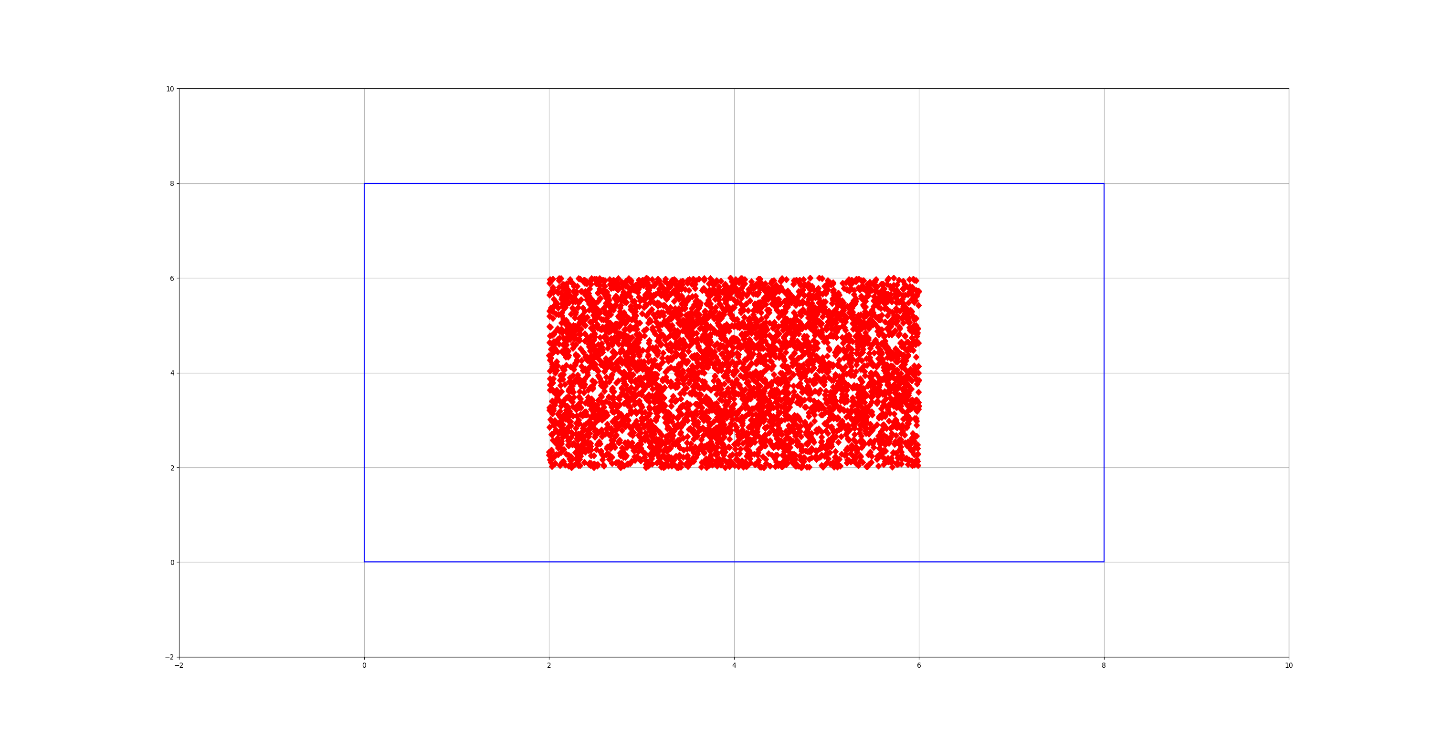


Figure 1: Initial distribution for 5000 neutrons generated between (2, 2) and (6, 6) cm, within bounds of (0, 0) and (8, 8) cm.

For each neutron generated, we sample a random direction from 0 to 2π, and a track length proportional to . To determine the type of reaction the neutron goes through, we generate a random value [0, 1) and compare it to . We run this model for each neutron until it is absorbed or leaked outside the initial bounds. Once every neutron path is complete, we sum the path lengths within each discretized cell of our grid to find the scalar flux distribution. Other tallies we keep include the number of neutrons leaked from each direction, and the number absorbed within the boundaries.

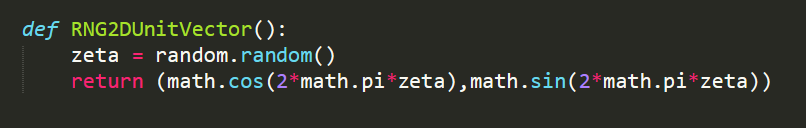
Mathematics:

The neutron transport problem we are modeling is listed below:

It has loss terms of leakage and collision, and gain terms of inscattering and source. Instead of looking at only one specific angle, our model looks at every angle, as otherwise any simulation would be difficult.

Algorithms:

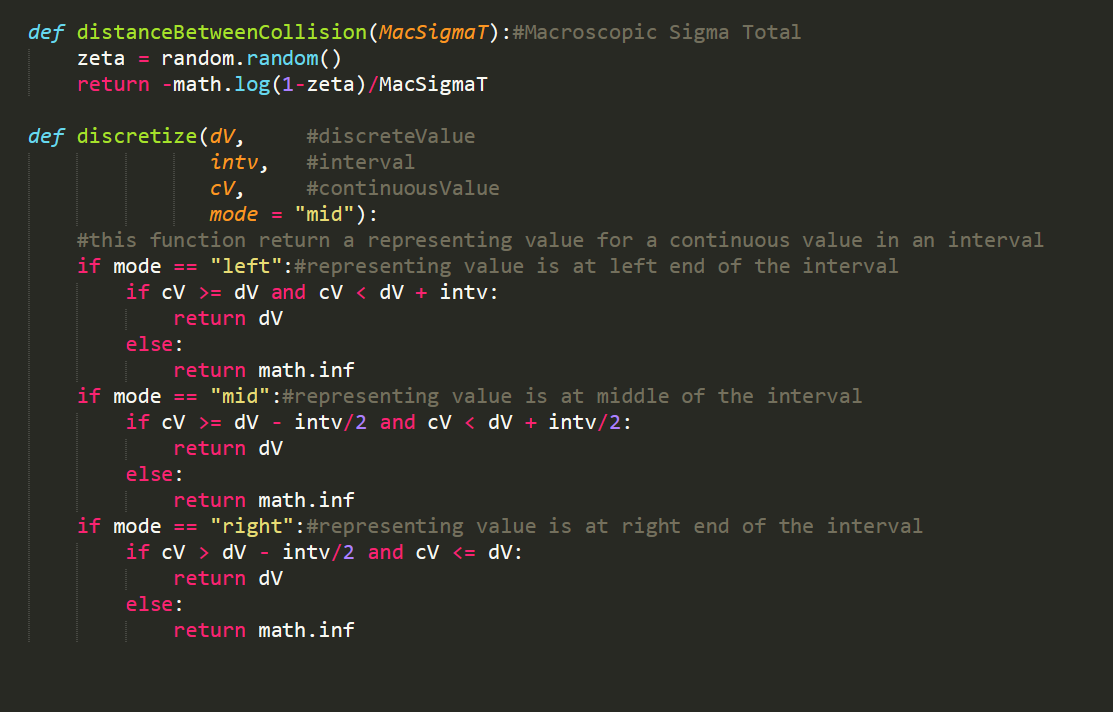
Directional sampling:



This function returns the x, y unit circle coordinates of a randomly generated angle from 0 to 2π.

(Vujic 13, p41)

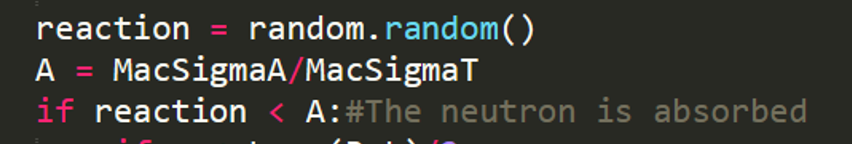
Track length sampling:



This is determined from the probability density formula for having an interaction at distance *s* being equal to , where *ΣT* is the total macroscopic cross section. We integrate the PDF to get the CDF, from which we can sample with a random number generator:

(Vujic 15, p15)

Collision sampling:



This is not a function; it is rather three lines from the body of our code. However, it is important to mention in order to sample a cross section, we take the ratio of that cross section over the total cross section and compare it to our random number *ξ*. If we have multiple cross sections, we would compare for *Σ2*. However, since we only have two interactions, we can say that if the neutron is not absorbed, it is scattered.

(Vujic 13, p37)

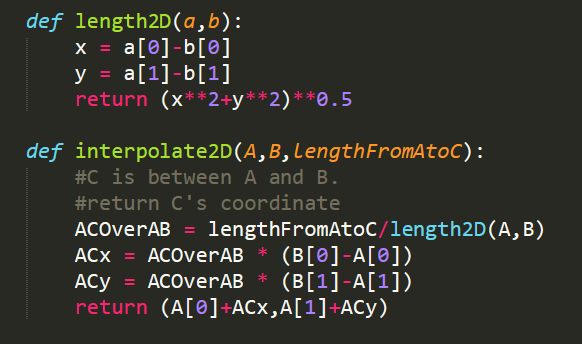
Rectangular MC simulation:

(see appendix [2] for code)

This body function instantiates leakage boundaries and tallies, and runs the simulation for *N* neutrons, until they are all absorbed or leaked. It outputs tallies for left right up down leaked neutrons, as well as absorbed neutrons. It also outputs neutron location at every single interaction a neutron undergoes.

(Vujic 15, p12)

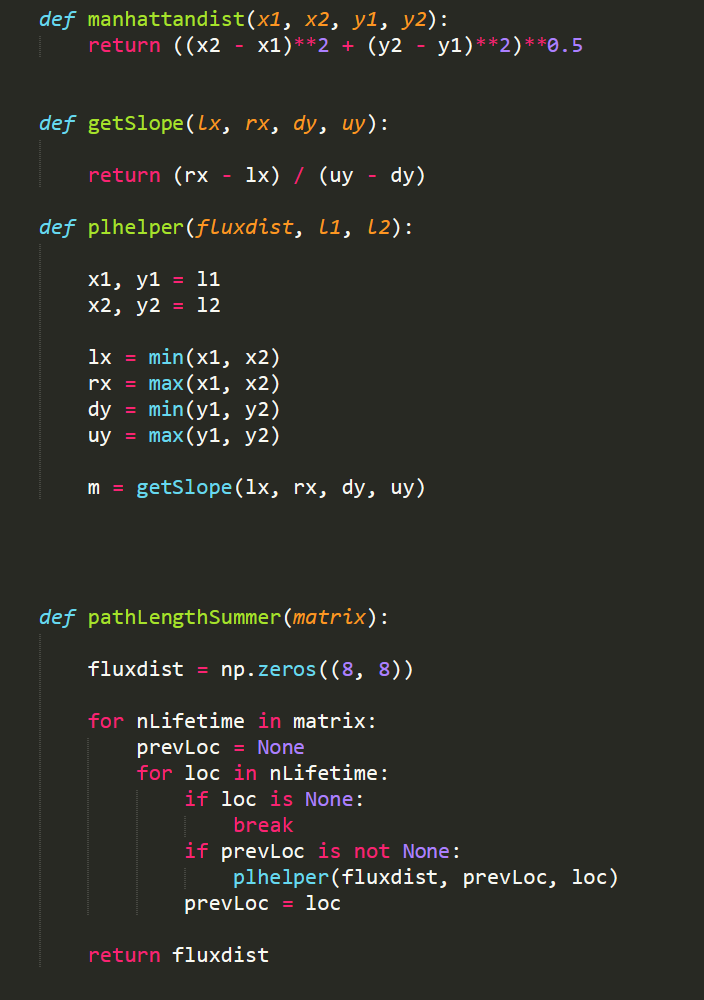
Flux helper functions:



length2D returns the Manhattan distance between two points.

interpolate2D returns the point directly between A and B.

We ended up using this to model neutron flux. Each point C adds one integer to the flux distribution. This mainly came from a disagreement between project partners. If I had more time I would have summed the path lengths within each grid, as I believe is the right method. However, it is 12:05AM so I choose to use the completed method which we have. Here is the incomplete code I had written for what I believe is the correct method:



The next step would have been to find the points along the grid which corresponded to the line *mx + y1.* With those points, we would take the Manhattan distance the line crosses in each grid point and add it to the *fluxdist* matrix. Once we did the same for every neutron, we would have an accurate flux distribution after normalization.

(Vujic 16, p20)

Flux calculation:

(see appendix [3] for code)

Instead of having a flux distribution matrix of path lengths, we have a flux distribution matrix of points between two collisions. This function returns the neutron flux at *t* microseconds after initial birth of neutron. It assumes neutron is moving at 2000m/s.

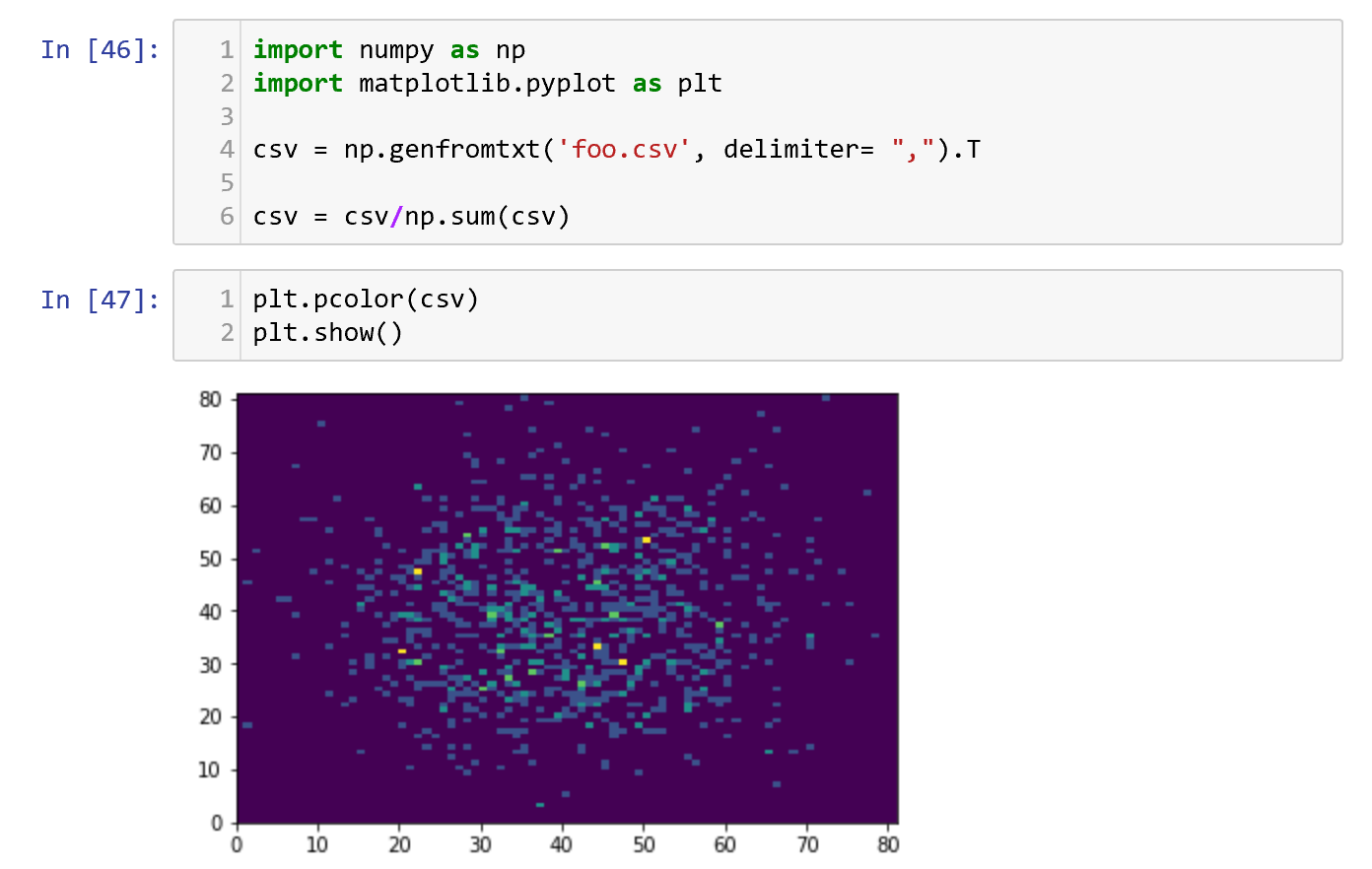
(Vujic 16, p20) 

Figure 2: Our flux distribution. I spent 30 min trying to figure out pyplot 3D plot but now it’s 1.

Code Use:

|  |  |  |  |
| --- | --- | --- | --- |
| Function: | Description: | Inputs: | Outputs: |
| RNG2DUnitVector | Samples random direction | None | Unit vectors *x, y* |
| distanceBetweenCollision | Samples random track length | *ΣT* | Distance *s* [cm] |
| Collision sampling | Samples a collision | *ΣA,ΣT* | Absorbed = True/False |
| Rectangular MC simulation | Runs simulation for *N* neutrons | Leakage bounds, *N* | Leaked L,R,U,D; Absorbed L,R,U,D |
| Flux calculation | Returns matrix of neutron flux | Neutron positions matrix | Neutron flux matrix |

Test Problems and Results:

We wrote a python script that takes in a matrix of every neutron position at every timestep and displays a pyplot animation of the position of each neutron at each timestep. A full cycle for 10000 neutrons can be found in the appendix.

At first, we saw that the neutrons in our code gravitated disproportionally towards the first and third quadrants. We thought this was due to an error in our direction sampling algorithm. However, it was a simple algorithm and everything turned out to be implemented correctly. On closer examination, we discovered we were incorrectly updating our neutron positions with “y = x + dy” instead of “y = y + dy”. After the fix, the neutron distribution seemed to be working as intended. This is apparent in appendix [1].

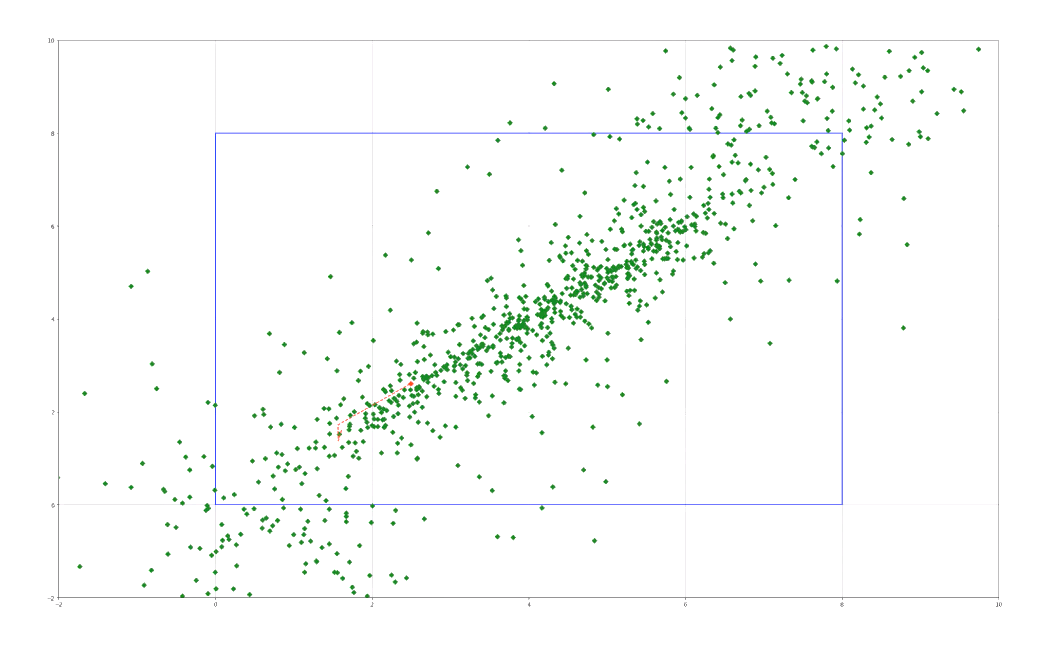


Figure 3: Imbalanced distribution of neutron direction

References:

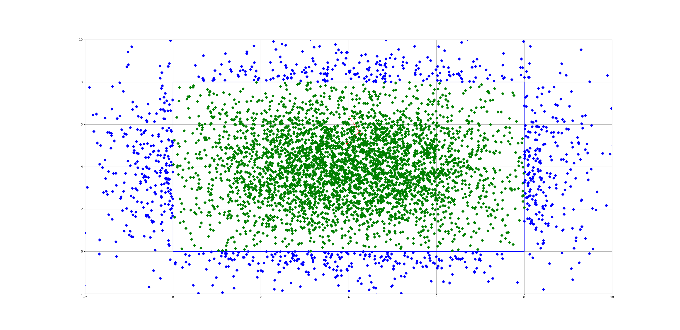
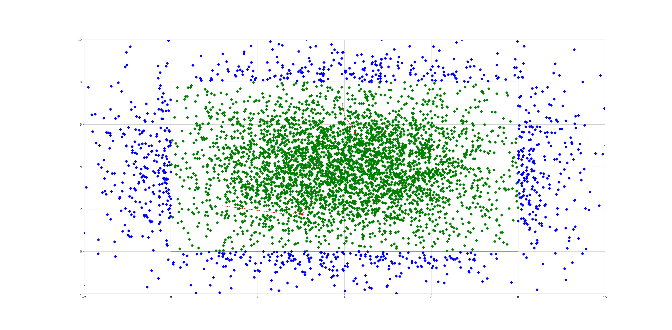
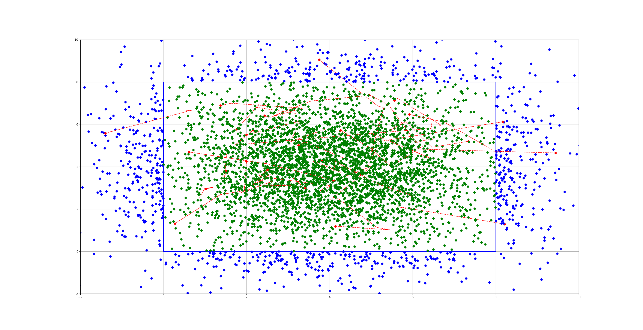
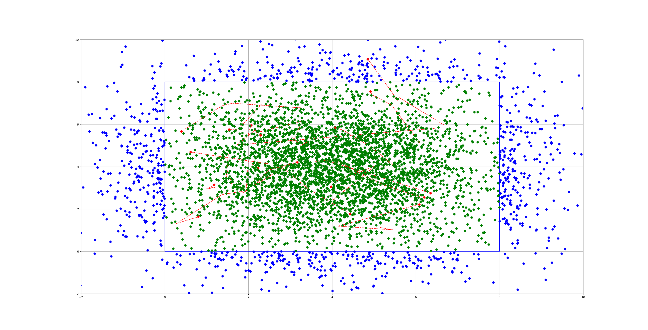
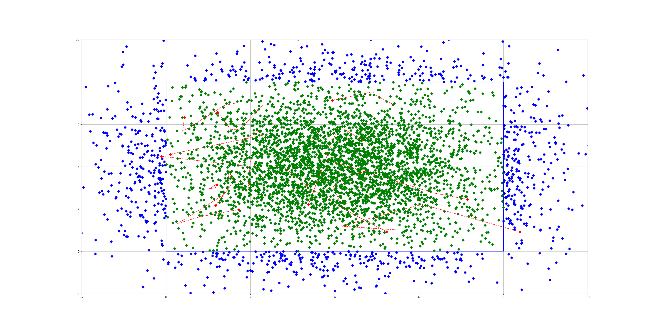
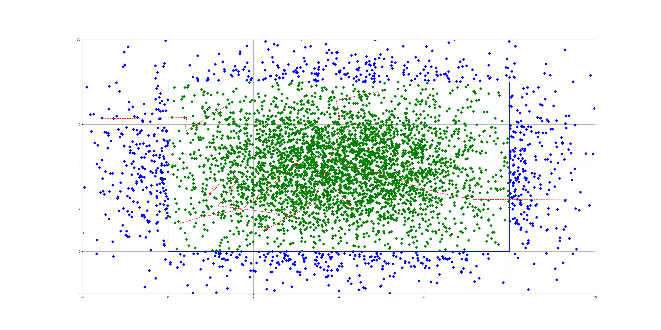
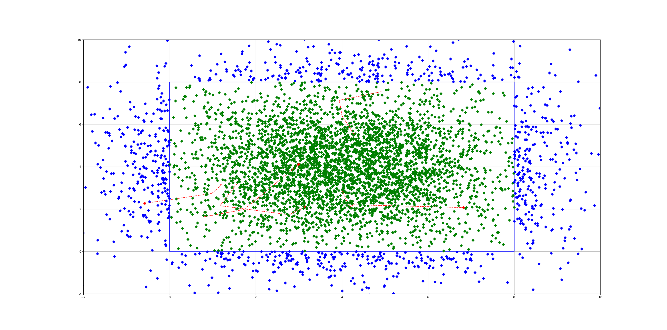
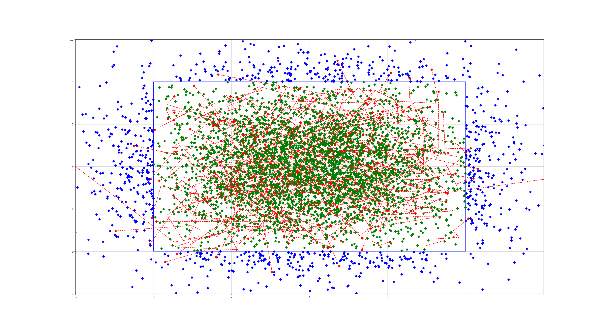
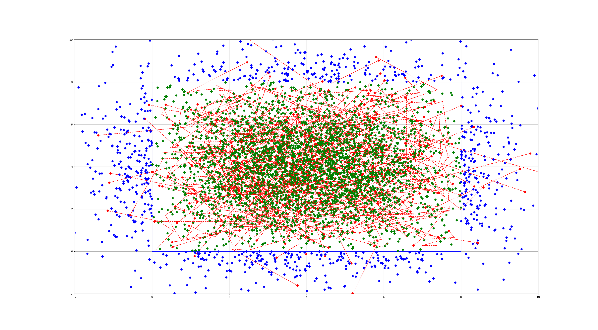
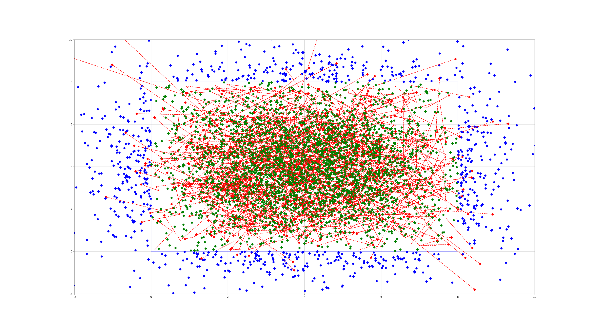
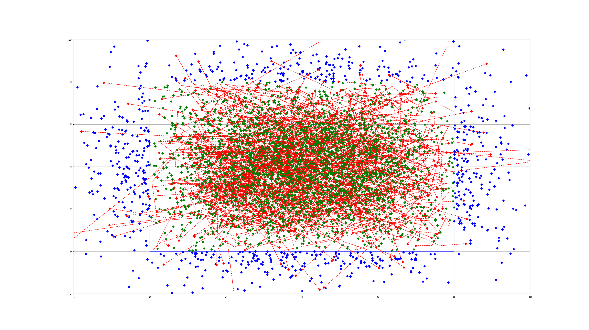
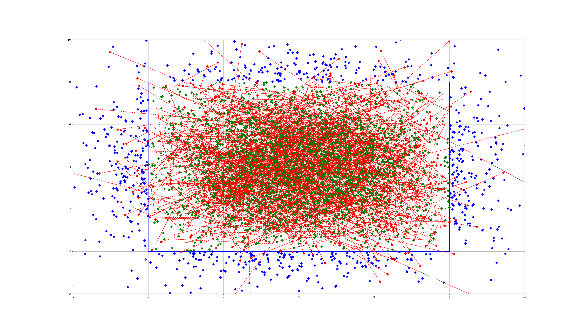
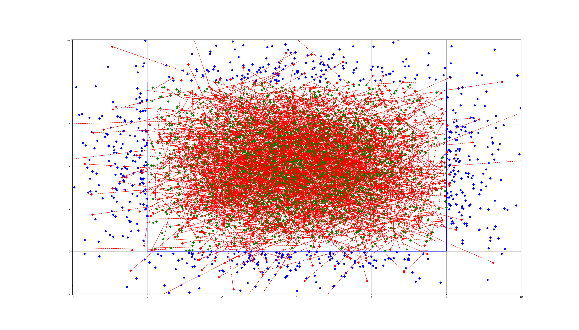
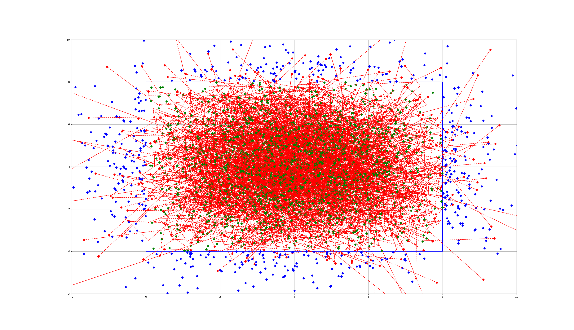
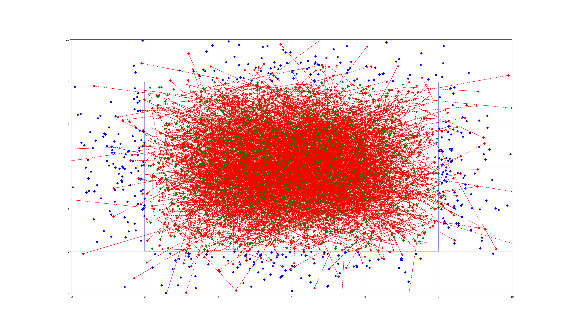
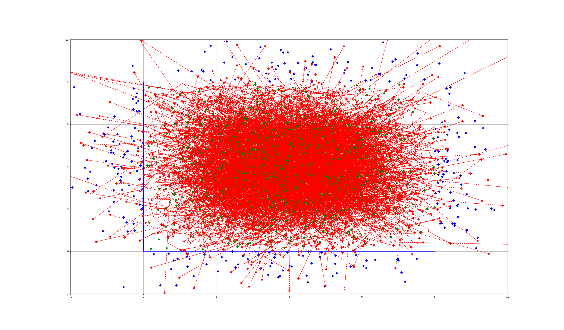
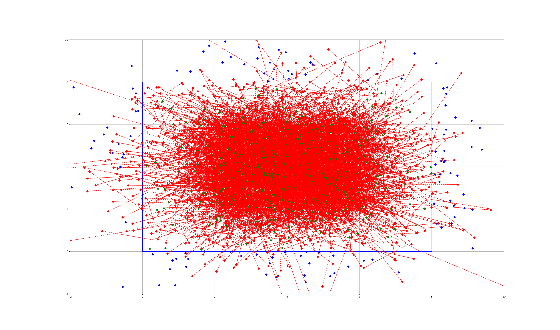
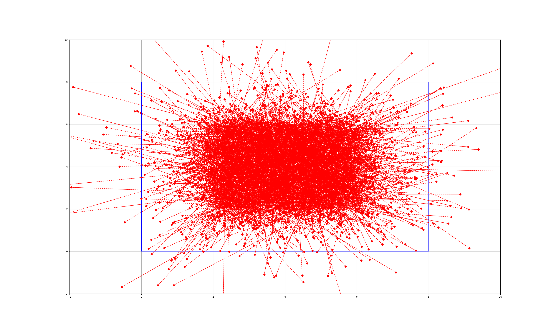
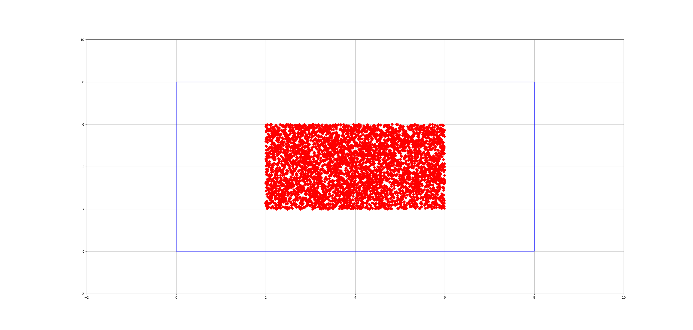
Vujic, Jasmina. *Lecture 13. Introduction to Numerical Simulations in Radiation Transport,* 2019.

Vujic, Jasmina. *Lecture 14. Introduction to Numerical Simulations in Radiation Transport,* 2019.

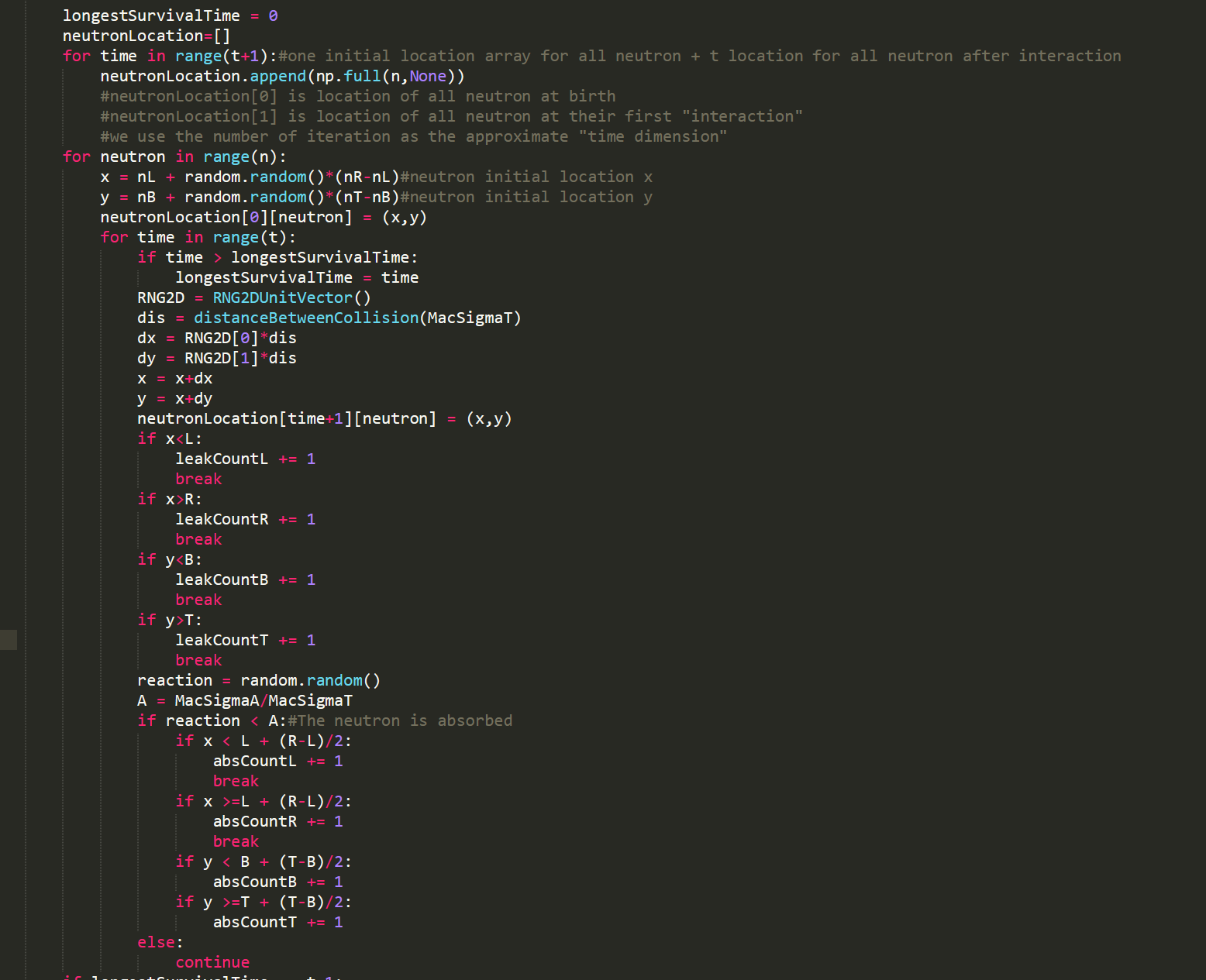
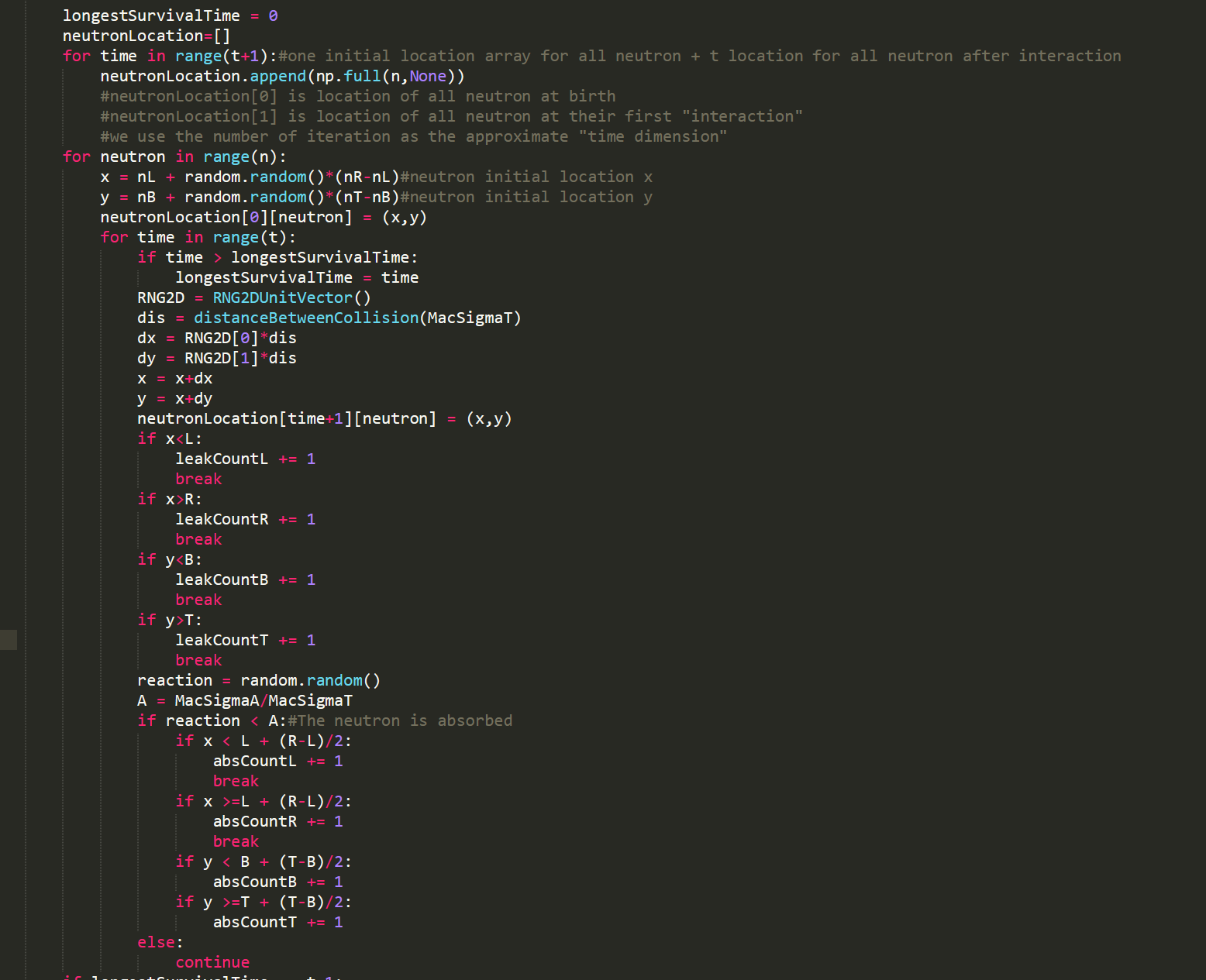
Vujic, Jasmina. *Lecture 15. Introduction to Numerical Simulations in Radiation Transport.* 2019.

Vujic, Jasmina. *Lecture 16. Introduction to Numerical Simulations in Radiation Transport.* 2019.

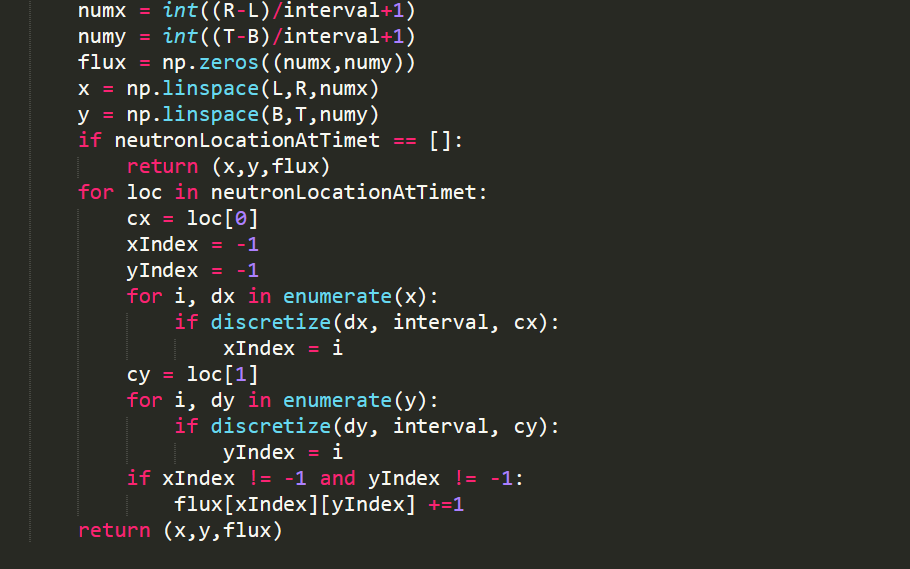
Appendix:



Appendix 1. Compilation of visualization data for 10000 neutrons, ordered left to right.



Appendix 2. Main body code for rectangular MC simulation

Appendix 3: Approximation code for flux distribution