Capstone Project Report

Introduction

The aim of this project was to effectively model a fission nuclear reactor using python. The general structure is as follows. Beginning with a set number of neutrons, a Monte Carlo simulation was run to determine the number of fission, scattering, absorption, and escaped neutron events. Determining which events happen utilizes the cross-sections of the different materials in the reactor. A cross-section is a measure of area and can be used to determine the mean free path or the probability of an interaction happening, dependent on the particular elements involved and the energy of a given neutron. Using the cross-sections, cumulative distribution functions can be found which are used in the simulation.

With the number of different events, several interesting physical properties of the reactor can be calculated. Primarily, whether the reactor produces as many neutrons as it consumes; i.e. is the reactor critical. Other results relate to the enrichment of the reactor, the implementation of control rods, manipulating the reflectivity and size of the reactor, as well as observing how many antineutrinos the reactor produces.

Implementation

Our Process

Reaching our final design took several iterations, as multiple hurdles had to be overcome before we reached a workable model. This process saw us first simulating a single neutron, then turning this model into a crude Monte Carlo. As we worked towards a more polished model, we split our code into several interacting modules. These modules would first spawn a set number of neutrons into the simulation, process their interactions based on their cross-sections (probabilities), and produce counts of each event which occurred. These three modules, in order, may be referred to as the 'neutrons' modeule (initialize_neutrons), the 'density' module (density_functions), and the 'escape' module (check_escape). Finally a fourth 'master' file would be used to bring each of these pieces together by initializing the reactor size, number of neutrons, the particular elements involved, and then running each of the other modules. This final file was labeled the 'reactor' module (reactor_function) in the final iteration.

The second significant update put in much of the work to create cohesion between the modules. It updated the escape module so that event counters could be reset within the module itself, and created a consistently functional reactor module. A visualization module (reactor_visualizations) module was also added, allowing for several graphical displays of input-output interactions. This ultimately helped us verify how realistic our reactor was, allowing us to visualize several known aspects of nuclear reactors as well as indicating places where there was room to improve our simulation. Many of the graphics from this visualization module will be shown in the results section below. Finally, upon noticing in our visualizations that our base reactor wasn't nearing criticality, we added a variation of our escape module that allowed for partial reflection (rather than escape), providing more realism as well as helping to demonstrate criticality.

Code Walkthrough

Neutrons Module

The neutron module creates a 'neutrons' function which takes as inputs the number of neutrons (count), the reactor radius, and the energy of each neutron (array-like). Firstly, a (3, 3, count) array is initialized with random values between 0 and 1. Each neutron was represented by a slice along axis 2 and encoded an (x, y, z) position, direction, and an energy. The first row's random values (0-1) were used to give each neutron a randomized spherical coordinate which were then converted to Cartesian coordinates. The second row indicated the direction (x, y, z) the particle would travel, and was converted to unit length. The neutron energies (an input to the original function) would be placed in the position (2,0), with the final 2 positions in the array unused.

Mixtures Module

By unpacking and processing a csv file, we receive the cross-sectional information on uranium-238 [1, 2], uranium-235, boron-10 [3], and heavy water [4]. When initiating a 'mixtures' function, we input percentage values of the total reactor volume to be taken up by each material. These volume percentages were converted to atomic percentages which was generally more useful for our calculations. These weighted densities allow us to show the probability of each event (fission, absorption, scattering) for both high- and low-energy neutrons. These new weighted cross-sections can then be used as inputs to our probability density function.

Density Module

The density function was somewhat simpler. Firstly, the probability density function takes the number of particles and total cross section as inputs, outputting a function of 'x'. The cumulative distribution function integrates over the pdf. By taking the inverse of the cdf and mapping onto it a random value (0, 1), we find the distance each particle will travel before experiencing an interaction. [5]

Escape Module (No Reflection)

The escape module creates a 1d array which, using the inverse cdf function, determines the distance each neutron will travel before an interaction. By moving the particle's position in the specified direction, it will receive a final position. If this position is outside the reactor, the neutron will have successfully escaped; otherwise it will perform a different event based on the probabilities taken from the mixtures module. In the case that the neutron scatters, its energy may be decreased and it will be run through the simulation recursively.

Reflective Shell

A variation of the escape module allowed neutrons to reflect off a wall rather than simply escaping. This was done probabilistically depending on a certain defined reflectivity. By making this change, our reactor was much more real-to-life and could reach a critical level.

Reactor File

The reactor file is the main file which gets run, calling each of the others as necessary. By having all the alterable variables in one place, it is very easy to run and see how changing inputs affect each of the simulated results.

Results

Following the completion of our reactor simulation, we coded several visualization functions focused specifically on generating graphs. These visualized how our simulation reacted to shifting input variables and can be broadly categorized into two groups: material and criticality graphics.

Material Graphics

Material graphics focused on varying the percentages of our material inputs and seeing how our counters reacted.

Looking first at uranium percentages, we are able to demonstrate the impact of enrichment on fission by making a graphic which ran our simulation from 1% enrichment (99% U-238, 1% U-235) to 100% enrichment. This graph showed us that the largest gains stopped at 20% but the number of fissions still continued to increase slowly. This matches quite well with real world use-cases, where the vertical lines on the graph separate natural uranium, reactor-grade uranium, and weapons-grade uranium.

Our second material graphic sought to observe the impact of a moderator on the amount of scattering within the reactor. By varying the percentage of moderator (heavy water) from 1% to 66% (higher percentages would result in a non-reactive fuel), with 20% enriched uranium, the increase was close to linear.

For the last materials graphic, we could see the impact of control rod insertion on the reactivity of a reactor; how power plants 'poison' their reactors to prevent meltdown. By varying the percentage of boron from 1-75% against fission counts, the graphic showed a rapid decrease in fissions with increasing quantities of boron.

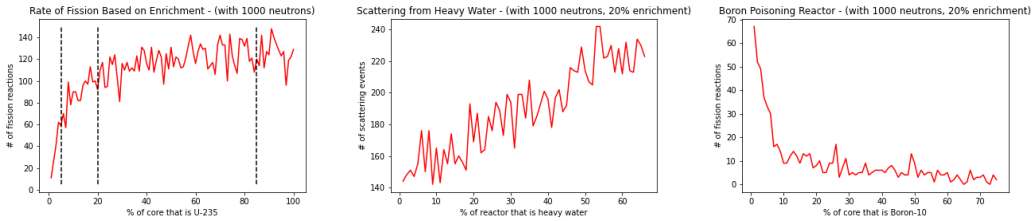

Figure 1: Enrichment, Moderator, Boron Poisoning (left to right)

Criticality Graphics

Our first criticality graphic used the model without reflectivity and sought to test if criticality was possible with a sufficiently large reactor core. By starting with 1000 neutrons, we looked for the value of reactor radius where the number of neutrons produced from fission exceeded this initial amount, given that the average number of neutrons produced from each fission was 2.5.

Following this same process with a 50% reflective shell showed that criticality could occur at a lower reactor radius if reflection was accounted for.

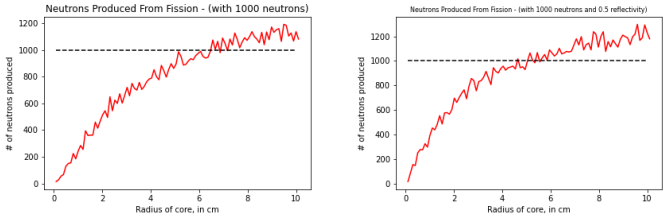

Figure 2: Criticality without (left) and with (right) reflection

The last criticality graphics illustrated the ways in which enrichment and reflectivity impact reactivity. The first graphic varied the reflectivity (rather than the radius) and graphed against the number of neutrons produced.

Similarly, a second graph varied the number of neutrons produced by the level of uranium enrichment.

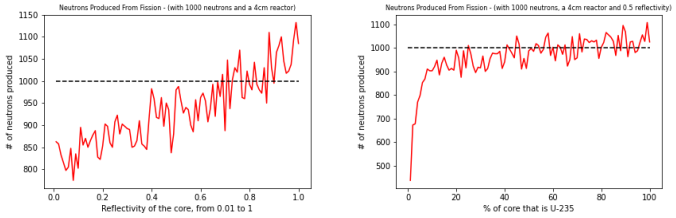

Figure 3: Criticality vs. reflectivity (left); Criticality vs enrichment (right)

Discussion

Looking at our graphical results, we see much to indicate the quality of simulation, particularly in its accuracy reflecting the mechanisms and characteristics of real-world reactors.

One particularly notable example is in the enrichment of uranium. Unenriched uranium, in the 1-5% range, is clearly much worse for fusion, improving quickly through the 5-20% reactor-grade enrichment. At higher levels of enrichment, the rate of return diminishes but is still non-negligible. This matches the idea that reactor-grade uranium is efficient while remaining cost effective. However for weapons, which intend maximum yield, it is sensible to use highly enriched uranium.

Our control rod graph shows, unsurprisingly, that greater rod insertions leads to a decrease in fissions. Excitingly, our reactor showed that 10% boron pushes fission rates to near-zero: very close to the 5-10% of volume used in real reactors to kill a reaction.

Our criticality graphs show that size, enrichment, and reflection all increase the potential for a critical reactor. This is necessary for energy production, but also demonstrates how dangerous reactors can be without mechanisms for moderation, leading to the possibility of going supercritical.

Concluding Remarks

Overall, this simulation was very effective in what we set out to achieve. We saw the expected trends that should be seen, despite various assumptions made during the process of programming the simulation.

Throughout our process, the use of matricies and function mapping rather than loops led to significant speed improvements. As a group our work on the physics, code, presentation, and write-up were well balanced, as we learned to work individually as well as combining our efforts. One area we could improve is working in parallel on different tasks by separating modules early. What we did instead was work on the ideas sequentially and separated only for reasons of organization, leading to a slower group workflow.

If we wanted a more accurate model, there are a few directions which could significantly expand the project's scope. One such task would be using neutrons that are born with some distribution across the 'fast' energy regime and having unique energy loss from different scattering events. This would involve new cross-sections as well as certain numerical integration problems due to the cumulative distribution function. Another improvement might be to include time-dependence which, along with computational power, would require an inclusion of components such as fuel burnup, temperature, and fission products. Finally, we might model real reactor and control rod geometries rather than assuming homogeneity.

References

[1] H. Kim, "Neutron cross section evaluation of u-238." [Online]. Available: https://t2.lanl.gov/fiesta2017/Talks/Kim.pdf

[2] O. Andersen, "Uranium 238." [Online]. Available: https://www.sciencedirect.com/topics/pharmacology-toxicology-and-pharmaceutical-science/uranium-238#:~:text=Uranium%20(U%2C%20atomic%20number%2092%2C%20in%20group%205%20of,is%20a%20naturally%20occurring%20actinide.

[3] D. Bogart, "Boron cross sections as - a source of discrepancy." [Online]. Available: https://ntrs.nasa.gov/api/citations/19660018445/downloads/19660018445.pdf

[4] J. A. E. Agency. [Online]. Available: https://wwwndc.jaea.go.jp/cgi-bin/Tab80WWW.cgi?/data/JENDL/JENDL-4-prc/intern/B010.intern

[5] OpenMC. [Online]. Available: https://docs.openmc.org/en/latest/methods/neutron_physics.html

[6] "Delayed neutrons." [Online]. Available: https://ec.europa.eu/programmes/erasmus-plus/project-result-content/fbcfee8c-7b46-4e39-a5eb-129ea94d51c7/CTU1_Delayed_neutrons_part_2_Experiment_procedure_for_students.pdf

[7] foxneSsfoxneSs, "Local (?) variable referenced before assignment." [Online]. Available: https://stackoverflow.com/questions/11904981/local-variable-referenced-before-assignment

[8] W. McDowell. [Online]. Available: https://slideplayer.com/slide/8428135/

[9] username4567, "Calculate inverse of a function–library." [Online]. Available: https://stackoverflow.com/questions/15200560/calculate-inverse-of-a-function-library

[10] Sep 2024. [Online]. Available: https://en.wikipedia.org/wiki/Neutron_transport