TALLINN UNIVERSITY OF TECHNOLOGY
School of Science

Siim Erik Pugal 214704YAFM

Simplified Modelling of Neutron Transport in Python

Master's Thesis

Supervisor: Marti Jeltsov
Ph. D.

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Loodusteaduskond

Siim Erik Pugal 214704YAFM

NEUTRONITE TRANSPORDI LIHTSUSTATUD MODELLEERIMINE PYTHONI BAASIL

Magistritöö

Juhendaja: Marti Jeltsov
Ph. D.

Tallinn 2024

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Siim Erik Pugal

04.01.2024

Supervisor’s Approval

The thesis meets the requirements for a master’s thesis.

Author: Marti Jeltsov

04.01.2024

Abstract

The Boltzmann transport equation is a mathematical equation used in physics to describe how particles, such as neutrons, move and distribute themselves in a medium. It is notorious for being one of the most difficult equations to be solved in nuclear physics.

This master's thesis explores two well-known numerical methods, which aim to find an approximate solution for the criticality value of a typical pressurised water reactor using the Boltzmann transport equation. Those solution methods stem from two main branches: stochastic and deterministic. From stochastic methods, Monte Carlo offers a way to model the inherent randomness a neutron takes inside the reactor's unit cell. The alternative approach, which uses a deterministic approach, is called the discrete ordinates method, which discretizes the number of angles the neutron might take, while taking part in neutron-nucleus reactions.

The Monte Carlo method is a statistical technique used to solve problems through random sampling. In the context of simulating particle behaviour, like neutrons in a nuclear reactor, the Monte Carlo method involves randomly sampling various events that particles can experience, such as fission, scattering or absorption. However, in order to simplify the neutron's free path calculations in different materials, a special method is applied to speed up this process: the Woodcock delta-tracking method. The general idea behind this process is to model so-called virtual collisions in addition to real ones in order to make calculations more efficient, while still capturing the essential characteristics of neutron transport.

The method of discrete ordinates, or S_N for short, involves discretizing the solid state angle into a finite set of directions using the Lebedev-Laikov quadrature rules in order to solve the k-eigenvalue problem. To achieve this, the steady state form of the Boltzmann transport equation is transformed from the integro-differential form into a system of linear equations. This system of linear equations is solved by the BiGCSTAB algorithm to approximate the neutron flux from the transport equation, which is then used to describe the criticality level of the reactor.

This thesis is meant to serve as a step-by-step tutorial on how to solve the Boltzmann transport equation using the Python programming language. The findings aid in advancing our understanding of neutron transport in nuclear reactors and have implications for reactor

safety analysis and design.

Annotatsioon

Neutronite transpordi lihtsustatud modelleerimine Pythoni baasil

Boltzmanni transpordivõrrand on matemaatiline võrrand, mida kasutatakse füüsikas erinevate osakeste, nagu näiteks neutronite, liikumise ja keskonnas jaotamise kirjel- damiseks. See on üks keerulisemaid võrrandeid, mida võib tuumafüüsikas lahendada.

Käesolev magistritöö uurib kahte populaarsemat numbrilist meetodit, mille eesmärgiks on leida ligikaudne lahend tüüpilise surveveereaktori kriitilisuse väärtuse jaoks kasutades Boltzmanni transportvõrrandit. Need lahendusmeetodid pärinevad kahest peamisest harust: stohhastiline ja deterministlik. Stohhastilistest meetoditest pakub Monte Carlo võimalust modelleerida neutronite liikumist juhuslikust vaatevinklist. Alternatiivne lahendusmeetod, mis kasutab deterministlikku lähenemisviisi, nimetatakse diskreetsete ordinaatide meetodiks. Selle idee on vaadelda lõplikut hulka erinevaid nurkasid, mida mööda neutron võib lennata, kui ta osaleb aatomituumaga seotud kokkupõrgetes.

Monte Carlo meetod on statistiline meetod, mida kasutatakse juhusliku valimi abil erinevate probleemide lahendamiseks. Simuleerides osakeste käitumist, nagu näiteks neutronite liikumist tuumareaktoris, hõlmab Monte Carlo meetod kindlate sündmuste modelleerimist juhuslike kokkupõrgete mõjul, milleks võib olla tuumade lõhustumine, hajumine või neeldumine. Selleks, et lihtsustada erinevates materjalides neutronite liikumisteekonna arvutusi, rakendatakse erilist Woodcock'i delta-jälitus meetodit. Selle protsessi üldine idee on lisaks reaalsetele põrgetele modelleerida ka niinimetatud virtuaalseid kokkupõrkeid, et muuta arvutuste kiirus tõhusamaks, kuid samal ajal säilitades põhilised omadused neutronite transpordi juures.

Diskreetsete ordinaatide meetod, lühidalt S_N, hõlmab kindlate nurkade diskretiseerimist, kasutades selleks Lebedev-Laikovi kvadratuurreegleid k-omaväärtus probleemi lahendamiseks. Selle arvutamiseks transformeetritakse Boltzmanni transportvõrrandi integraal-diferentsiaalkuju suureks lineaarvõrrandite süsteemiks. Selle võrrandisüsteemi lahenend leitakse BiGCSTAB algoritmi abil, mis arvutab välja neutronite voo, millega on võimalik leida reaktori kriitilisuse taseme väärtus.

See lõputöö on mõeldud olema samm-sammuline õpetus, kuidas lahendada Boltzmanni transpordivõrrandit Pythoni programmeerimiskeele abil. Saadud tulemused aitavad edendada meie arusaamist neutronite transpordist tuumareaktorites, millel on omakorda mõju reaktori ohutusanalüüsile ja projekteerimisele.

List of Abbreviations and Terms

BiCGSTAB	BiConjugate Gradient STABilized (method)
CSV	Comma Separated Values
DOM	Discrete Ordinates Method
GENDF	Generalised Evaluated Nuclear Data File
GPU	Graphical Processing Unit
GXS	Generalised (or Group) XS (Cross Sections)
HDF	Hierarchical Data Format
IAEA	International Atomic Energy Agency
MC	Monte Carlo
MOC	Method of Characteristics
PWR	Pressurised Water Reactor

Table of Contents

List of Figures

Introduction

Radiative transport, also known as radiation transport, refers to the movement of radiation through a medium. It plays a role in fields such as reactor physics, nuclear engineering, high-energy particle physics and astrophysics.

In reactor physics and nuclear engineering, radiative transport is used to describe how particles behave as they travel through a medium and are influenced by processes such as absorption, emission and scattering. In high-energy particle physics, radiation transport is crucial for understanding how particles behave in accelerators and during collisions. These collisions generate energetic particles that can penetrate matter. Analysing how these particles interact and move through materials is essential for designing particle detectors and interpreting data.

Radiation transport is also vital for shielding. Whether its facilities, nuclear power plants or rockets that go to space. Shielding plays a role in protecting both humans and sensitive equipment from harmful radiation. By studying radiation transport we can identify the materials and configurations that minimise radiation exposure and ensure the safety of the individuals and machinery involved.

In the field of astrophysics, radiation transport forms the foundation for understanding the behaviour of celestial objects. Stars, for example, release radiation in a range of wavelengths ranging from radio waves to gamma rays. Through the study of how this radiation propagates through space we acquire knowledge about the makeup, temperature and behaviour of stars and galaxies.

In the realm of medical imaging, radiation traverses the domains of X-ray imaging, computed tomography and positron emission tomography. These imaging techniques hinge upon the interplay between radiation and the human body to compose intricate images of internal structures. Through meticulous manipulation of radiation transport and astute analysis of its interactions, medical professionals can diagnose ailments, steer surgical procedures and gauge the efficacy of treatments.

Furthermore, in climate science, radiation transport also assumes a pivotal role, specifically in understanding climate change and the equilibrium between the influx of solar energy and the efflux of terrestrial energy. The Earth absorbs solar energy in the form of radiation.

This energy is assimilated, reflected and re-emitted by the atmosphere, clouds and the Earth's surface, which gives us an understanding of climate patterns and the ability to predict future weather transformations.

In this research, we have chosen to focus our simulations on a typical pressurised water reactor (PWR) as it represents one of the most prevalent and extensively studied types of nuclear reactors. PWRs are deployed widely across the globe. The publicly available data on the materials necessary to build this type of reactor makes it a prime example to understand the fundamentals of reactor physics. It is essential to note that the theoretical framework and methodologies employed in this study are adaptable and small adjustments can be made to extend the applicability of our simulations to different reactor types, such as fast sodium reactors or other innovative designs. Our approach can be easily modified to study different types of reactors, making it possible to investigate a variety of nuclear technologies.

Radiation transport truly embodies a multifaceted concept that encompasses an extensive array of disciplines beyond nuclear engineering. To understand it, we must consider the rudimentary principles of particle interaction and radiation transport as it traverses through matter. Numerous methods are available that encompass both deterministic and stochastic approaches. However, within the confines of this thesis, we focus on two of the most prominent methods for simulating neutron transport in order to find the criticality value for a typical PWR.

1. Theoretical Background

When we talk about the cross-section of a neutron, we do not exactly envision cutting the target nucleus in half and observing which section of the nucleus the neutron hits. Instead, we mean that it is an effective target area that quantifies the likelihood of a certain interaction between the incident neutron and the target nucleus.

Modelling neutron-nucleus interactions involves separating these reactions into three main types: scattering, capture and fission events. Each reaction has a specific probability of occurring. In other words, it is possible to classify different reaction outcomes into separate regions of the nucleus, meaning that if the nucleus were to hit a certain area of those "cross-sections", it would result in a specific event. The larger the region, the higher the probability of such an event occurring. In reality, this is not entirely true. A nucleus does not actually have well-defined cross-sectional areas. This is because nucleons (protons, neutrons, electrons) do not always act as particles and can also act as waves. For simplicity's sake, we assume that these well-defined areas exist.

1.1 Scale of a cross-section

The geometrical diameter of a nucleus is incredibly small, a little under two barns for a ^{235}U nucleus. The unit "barn" is used to measure the cross-sectional area. One barn is equal to 10^{-28} m^2. It is a very small area to hit with neutrons, but as it turns out, it is not as difficult as it may initially seem.

The unit name barn has an amusing origin story. During the Manhattan Project, scientists needed a way to describe the probability of a neutron hitting the nucleus. They wanted a unit that conveyed the idea of a large target, something that would humorously emphasise the poor aim needed for that. Legend has it that during a conversation between the physicists Marshall Holloway and Richard Baker, they were discussing the seemingly elusive nature of hitting a nucleus with a neutron. They suggested using the term "barn" as a unit of measurement. It is a nod to the American idiom "couldn't hit the broad side of a barn". That is, if a nucleus has a large cross-section, a neutron has a better chance of hitting the metaphorical barn. [1]

1.2 Microscopic Cross-section

A nucleus has four sections that classify it. This is called the microscopic cross-section. For reference, let us examine the different areas of a ^{235}U atom. The geometrical cross-section is actually the smallest of the four and is mainly the result of elastic scattering events. Next comes the scattering cross-section, which has a relative size of approximately 10 barns. The scattering cross-section is mainly concerned with elastic and inelastic scattering events. After that comes the capture section, which has a rough size of 99 barns. A capture event is usually characterised as the nucleus absorbing the incident neutron and emitting some radiation in return. Finally, the largest of them all is the fission cross-section with a relative size of 583 barns. This last section is what we are mainly concerned with and as you can see from Figure 1, it is not actually the hardest task to get a neutron to hit the target nucleus. [2]

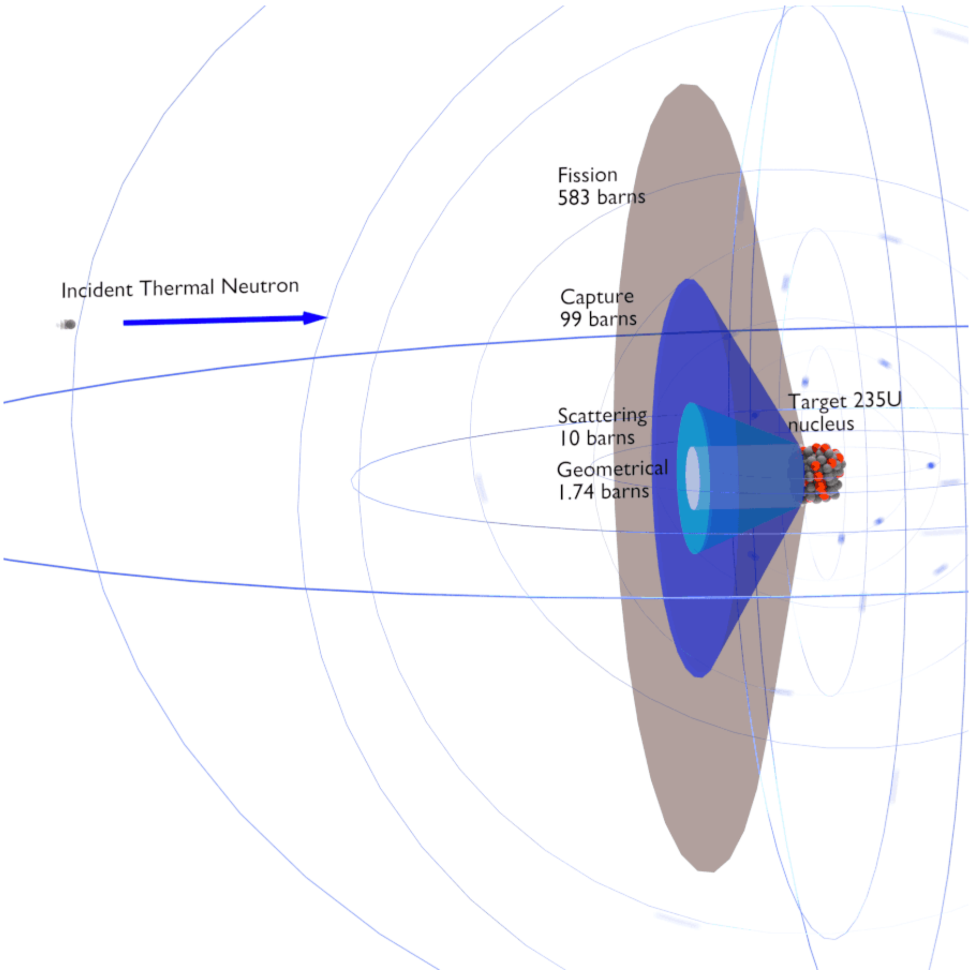


Figure 1. Various microscopic cross-sections of ^{235}U and an incident thermal neutron. [2]

It is imperative to recognize that the conceptualization of these cross-sections is rooted in a model. Although this model may seem unconventional, it effectively merges the ballistic perspective of neutron movement with the quantum mechanical properties inherent in the nucleus. Therefore, the concept of microscopic cross-sections arise as an artefact of this model, offering a unique yet functional framework for understanding neutron-nucleus interactions. The microscopic cross-section is denoted as σ. [3]

1.2.1 Total Cross-section

In general, nuclear cross-sections can be measured for all possible interaction processes together. In this case they are called total cross-sections (σ_t). The total cross-section is the sum of all the partial cross-sections such as:

- elastic scattering cross-section (σ_s)
- inelastic scattering cross-section (σ_i)
- absorption cross-section (σ_a)
- radiative capture cross-section (σ_γ)
- fission cross-section (σ_f)
- production σ_p = (n, p) reaction cross-section
- σ_α = (n, α) reaction cross-section
- ...

$$\sigma_t = \sigma_s + \sigma_i + \sigma_\gamma + \sigma_f + \dots$$

1.3 Classification of free neutrons

When classifying different neutrons on the basis of their kinetic energies, we can distribute them into at least 10 different groups. [4] In reactor physics, we are mainly concerned with three main groups of neutrons: thermal, resonance and fast.

Thermal neutrons are characterised by their lower energy state, which hover around 0.025 eV and 1 eV. For reference, a neutron energy level of 0.025 eV corresponds to an average neutron population at around 20 °C, with an average molecular speed of roughly 2 km/s based on the Maxwell-Boltzmann distribution. The average speed of neutrons for 1 eV are around 15 km/s. Thermal neutrons prove particularly effective in instigating nuclear fission reactions due to their heightened likelihood of being absorbed by fissile isotopes like ^{235}U, which is the common fuel for PWRs. This is because thermal neutrons exhibit the highest cross-section for absorption. [4]

Resonance neutrons are special. At certain energies, cross-sections can surge to over 100 times the baseline, with neutron capture significantly surpassing the likelihood of fission. It is crucial for thermal reactors to quickly overcome this range of energy, ensuring operation with thermal neutrons to enhance fission probabilities. The energy range for resonance neutrons is between 1 eV and 1 keV. [4]

Conversely, fast neutrons boast significantly higher energy levels, ranging from 1 keV to 1 MeV. These neutrons are commonly generated in fission reactions or in nuclear reactions with elevated energy levels. Fast neutrons are favourable in fast reactors, which use a different fissile isotope like $^{239}\mathrm{Pu}$. Despite that, uranium can also be used for fast reactors, but it would mean that far higher levels of enrichment are necessary for uranium to be a viable option. [4]

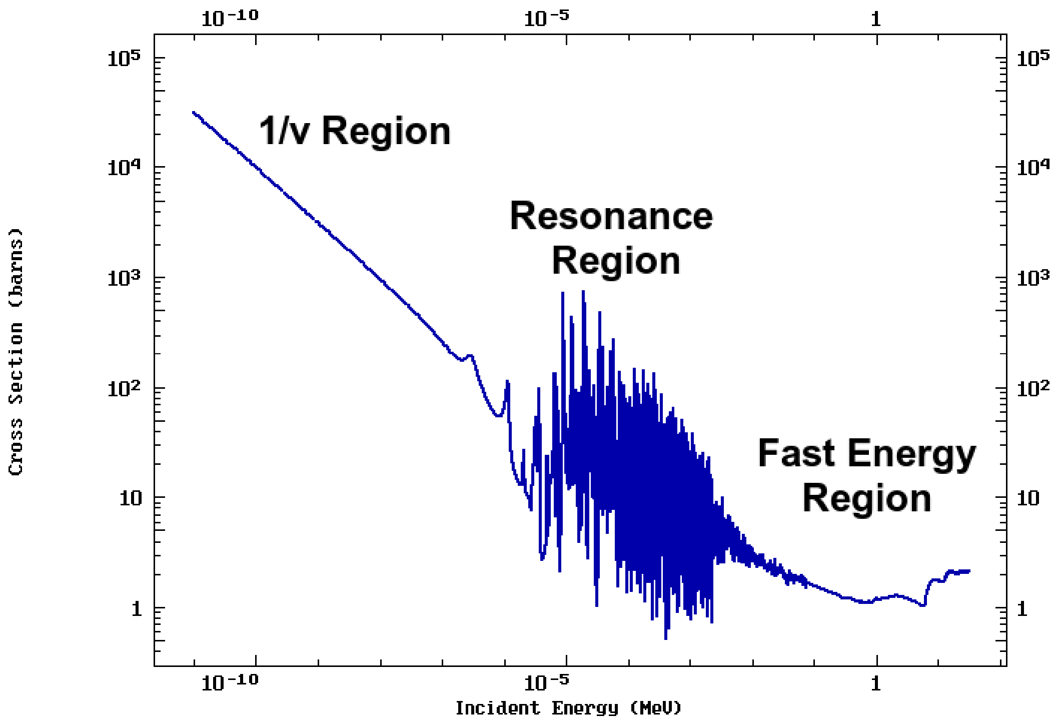


Figure 2. Microscopic cross-section of total fission for $^{235}\mathrm{U}$ with three main energy regions.

In general the neutron cross-sections can be divided into three main regions as a function of energy. This energy is the instant energy of the neutrons that interact with the nucleus. The three main energy regions of the cross-section are, in order of increasing energy:

1. $1/v$ Region
2. Resonance Region
3. Fast Energy Region

1.3.1 $1/v$ Region

Also known as the thermal energy region, the $1/v$ region is the first region, starting at the left-hand side of the cross-section and energy graph. It is a fundamental concept in the study of nuclear reactions, particularly those involving slow or thermal neutrons.

Thermal neutrons are neutrons that move relatively slowly and are much more likely to be captured by a nuclei. In this region, however, the cross-section is very large and mostly populated with slow neutrons. This indicates that there is a high probability of the incident particle being absorbed by the target nucleus. The shape of the cross-section in this region is a broad, smooth curve that peaks at low energies. [2]

At low energies, the cross-sections follow a $1/v$ distribution, where v is the velocity of the neutron. Using the definition of kinetic energy, we can ensure that this energy distribution is also equivalent to $1/\sqrt{E}$ distribution:

$$E = \frac{mv^2}{2} \to E \sim v^2 \to \sqrt{E} \sim v$$

$$\sigma_a \sim \frac{1}{v} \sim \frac{1}{\sqrt{E}}$$

While limited to the absorption cross-section in the $1/v$ region, the model proves exceptionally valuable. With a minimum of two measurements of cross-section values, one can establish a trend line and extrapolate across a considerably extensive range. This quality enhances its practicality and makes it a powerful tool for analysis within the specified region. [2]

1.3.2 Resonance region

The resonance region appears to be erratic and irregular because it consists of a large number of narrow, closely spaced energy levels. There are two resonance regions: resolved and unresolved. The resolved resonance region has well-defined, isolated energy levels that are easily distinguishable from one another, whereas the unresolved resonance region has overlapping and closely spaced energy levels that cannot be resolved into separate lines.

Resonance peaks manifest in neutron-nucleus cross-sections due to the quantum mechanical nature of nuclear forces. These nuclear reactions represent transitions between distinct quantum states or energy levels. The nuclear reaction rate is directly proportional to its specific cross-section. A resonance with an independent cross-section can be effectively characterised using the single-level Breit-Wigner equation, derived from the non-relativistic Schrödinger equation. [5]

The most significant cross-sections occur at such neutron energies, which result in the creation of long-lived states within the compound nucleus[1]. These specific energies, which lead to the formation of compound nuclei, are known as nuclear resonances and are commonly observed in the resonance region. In general, the widths between these resonances peaks decease as the energies increase. At higher energy levels, the widths may become comparable to the distances between resonances, making it challenging to observe further resonances. The total width of a compound nucleus exhibits an inverse relationship with its lifetime. Consequently, narrow resonances are associated with capture, whereas broader resonances result from scattering. [2]

1.3.3 Fast Energy Region

The radiative capture cross-section rapidly declines to extremely low values at energies beyond the resonance region. This sharp decrease is attributed to the formation of the compound nucleus in higher excitation states. In these heightened states of excitation, there is an increased likelihood that a neutron, through collisions with other nucleons, acquires energy surpassing its binding energy[2] within the nucleus. This heightened probability leads to a dominance of neutron emission over gamma decay. Furthermore, at elevated energies, inelastic scattering and the $(\mathrm{n}, 2\,\mathrm{n})$ reaction become more likely, diminishing the occurrences of both elastic scattering and radiative capture. [2]

1.4 Macroscopic Cross-sections

As we know at this point, neutrons interact with atoms in various ways. Sometimes they collide head-on, while other times they just scatter off. However, when we tackle actual real-life problems like shielding, fuel burnup or criticality calculations, there is no point in looking at individual interactions between neutrons and nuclei. Rather, our focus must shift toward an ensemble of particles capable of engaging with neutrons. Here, we introduce a new concept: *the macroscopic cross-section*.

It is essential to differentiate between microscopic and macroscopic cross sections. The microscopic cross-section represents the effective target area of a single nucleus for an

[1]A short-lived, highly excited and unstable state of an atomic nucleus that forms temporarily during a nuclear reaction.

[2]The energy required to disassemble an atom's nucleus into its constituent elements, namely protons and neutrons.

incident particle and is measured in surface area (or m^2 or cm^2). On the other hand, the macroscopic cross-section represents the effective target area for all nuclei contained in the volume of a certain material. The units for this are given in cm^{-1} and we shall explain why it has this type of unit.

Let us look at an example. Say we have a slab of fissionable material. This material has some surface area A and thickness d. If we bombard this slab with a swarm of neutrons, we want to know how many reactions occur or, more specifically, what is the reaction rate for the given material. The density of nuclei within the material dictates the reaction rate. When incoming neutrons traverse the empty spaces between nuclei without interaction, no reactions occur, resulting in a subdued reaction rate. If the individual nuclei in the slab interact with an incident neutron, only then will a nuclear reaction occur.

Introducing a new value A', which can be defined as the cross-sectional area of this slab that contains N nuclei per unit volume. Recalling from earlier, σ is the cross-section of the individual nuclei. The macroscopic cross-section can be defined as follows:

$$\Sigma = N\sigma. \tag{1.1}$$

As you can see N has units of $nuclei/\mathrm{m}^3$ and σ is measured in m^2. Using this, we get a unit for the macroscopic cross-section to be $1/\mathrm{m}$ or m^{-1}. To describe A', we can use this equation:

$$A' = n(Ad)\sigma = nV\sigma \tag{1.2}$$

The multiplication between A and d can also be seen as the volume V of the slab. This relation represents the total cross-sectional area of all nuclei of the slab. If we have a case where the area of the slab is much greater than the area of the macroscopic cross-section, $A \gg A'$, then this means that little collisions will occur and most of the particles will pass freely through the slab. However if $A \approx A'$, meaning we have a very high concentration of nuclei and many collisions will take place with a high probability.

This however offers us new insight into how the microscopic cross-section can be defined as well. Let us say we have a value R_0 that describes the number of particles that strike the plate every second. Using the relations we described previously, let us define a new value R, which is the nuclear reactions per second.

$$R = R_0\frac{A'}{A} = R_0\frac{N(Ad)\sigma}{A} = R_0 N d\sigma \tag{1.3}$$

We shall use this to define the microscopic cross-section:

$$\sigma = \frac{R}{R_0 N d} \tag{1.4}$$

The denominator of this equation can be defined as the number of collisions between the particles and the slab per unit time per unit area. But the main takeaway is that the larger the microscopic cross-section, the more likely we have more reactions per second and the larger is the total cross-section for all of the nuclei.

1.4.1 Mixtures and Molecules

In the previous section we described an example where we had a material that only consisted of pure fissionable material. However, almost all substances are never this pure and are most likely made up of a combination of chemical elements and compounds, each having different atomic densities and cross-sections.

This is why it is necessary to determine the macroscopic cross-section for each isotope and then sum up all the individual macroscopic cross-sections. Here are three main points to keep in mind when calculating the macroscopic cross-section for mixtures. Firstly, we can define the number of molecules per cubic centimetre in the material of density ρ and molecular weight M as

$$N_i = \rho_i \cdot \frac{N_A}{M_i}, \tag{1.5}$$

where N_A is Avogadro's number. The relation N_A/M is the number of molecules in one gram of the mixture. [6] Secondly, we can use the previous equation to define the equation for the mixture for n elements in the macroscopic cross-section:

$$\Sigma_{\text{mix}} = \sum_{i=1}^{n} N_i \cdot \Sigma_i. \tag{1.6}$$

Thirdly and most importantly, it is crucial to remember that the macroscopic cross-section of molecules is not always equal to the sum of the cross-sections of its individual nuclei due to anomalies caused by chemical binding energy. For example, the cross-section of neutron elastic scattering of water exhibits anomalies for thermal neutrons. [6] This occurs because the kinetic energy of an incident neutron is of the order of magnitude or less than the chemical binding energy; therefore, the scattering of slow neutrons by water (H_2O) is greater than that of free nuclei ($2\,H + O$).

1.5 Chain Reaction Modulation and Structural elements

It remains crucial to modulate the pace of the chain reaction to prevent it from accelerating too swiftly or decelerating too slowly. In a PWR, an excess of fast neutrons may lead to reaction depletion, whereas an overabundance of thermal neutrons could potentially induce a meltdown.

In pressurised water reactors, the governance of these reactions is orchestrated through the deployment of control rods, fashioned from materials such as boron or cadmium, which are adept at neutron absorption. Particularly, during periods of low power, when thermal feedbacks are minimal, control rods play a pivotal role in modulating the reaction rate. However, as the reactor attains criticality and thermal feedbacks become more pronounced, this complex interaction changes. At this point, the reactor, propelled by strong negative feedback mechanisms, desires to maintain criticality. The modulation of the reaction rate is then predominantly influenced by these feedbacks, with control rods assuming a crucial role in setting the core's outlet temperature. [7]

These materials decrease the rate at which the reaction occurs. Moreover, nuclear reactors incorporate various coolants, such as water or gas, to facilitate the dissipation of heat from the reactor core, thus maintaining optimal temperature and pressure levels. The configuration and operation of the reactor are meticulously overseen to maintain equilibrium between the chain reaction rate, the heat emanating from the reaction and the reactor's cooling capacity. [8]

In addition, significant consideration should be given to the selection of the structural material for fuel rod assembly lines. Zirconium stands as a favoured choice owing to its commendable attributes, including low neutron absorption and robust corrosion resistance. Another sought-after option is the exotic M5 alloy, a zirconium-based composite infused with trace amounts of niobium, tin and iron. It delivers enhanced strength at elevated temperatures and resistance to creep, making it suitable for demanding nuclear applications. Steel, too, merits consideration for its formidable structural prowess, albeit it contends with a heightened susceptibility to corrosion-induced impairments. [9, 10]

1.6 Doppler Broadening and Self-Shielding

At higher temperatures, the particles in a substance move faster and collide more often. This in turn gives rise to a phenomenon known as Doppler broadening. Fundamentally, it alters the way the nuclei are perceived by the neutrons, impacting the energy of the emitted or absorbed radiation.

Doppler broadening of resonances is a crucial phenomenon that enhances reactor stability. It plays a significant role in determining the fuel temperature coefficient, which represents the reactivity change per degree shift in the fuel temperature. This coefficient, often known as the prompt temperature coefficient, due to the immediate changes it causes on fuel temperature. In thermal reactors, the Doppler broadening, which contributes to this coefficient, is largely responsible for its behaviour and it also makes a notable impact in fast reactors. Notably, the prompt temperature coefficient tends to be negative in most thermal reactors. [2]

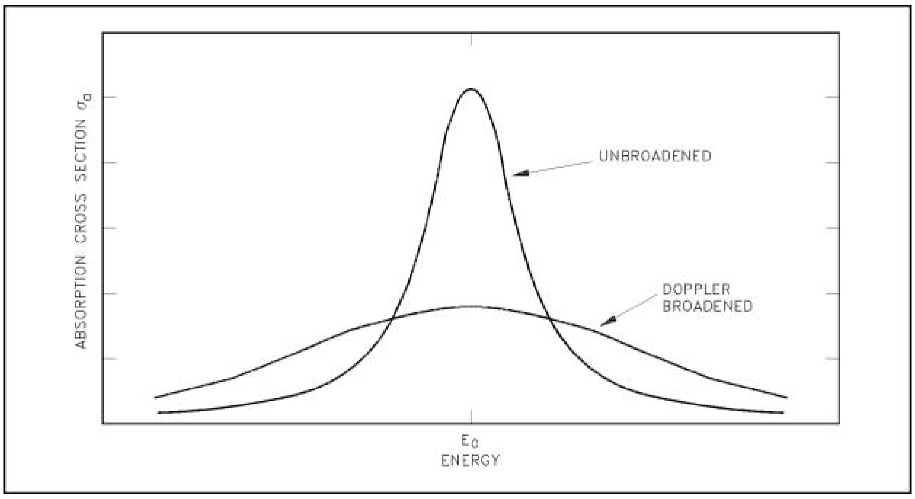


Figure 3. The Doppler effect graph. The Doppler effect improves reactor stability. Broadened resonance (heating of a fuel) results in a higher probability of absorption, thus causing negative reactivity insertion (reduction of reactor power).

In certain situations, even if the material's cross-section remains the same, the number of absorption reactions can drop significantly. This is often called resonance self-shielding and it also helps make the reactor more stable. Raising the temperature from T_1 to T_2 widens the spectral lines of resonances. While the total area under the resonance remains constant, the broadening of spectral lines leads to a rise in the neutron flux in the fuel, $\psi_f(E)$. Consequently, this increase in neutron flux contributes to higher absorption as the temperature rises. [2]

2. Neutron Transport Theory

Neutrons play a pivotal role in nuclear power generation, nuclear weapons, shielding and various other applications. Understanding how neutrons move and interact with materials is crucial for designing safe and efficient nuclear systems.

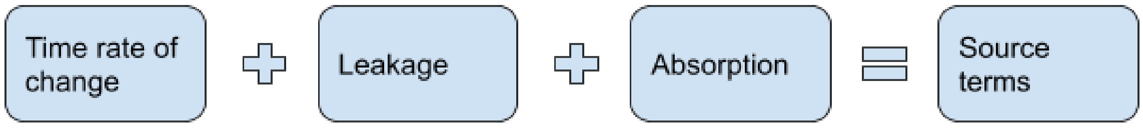
2.1 History

Neutron transport theory is mainly concerned with the transport of neutrons through various media. In reactor physics it is used to calculate the reaction rate of a nuclear reactor. Deriving the fundamental equation from the conservation laws is not very complicated. However, solving the said equation is quite notorious for how complicated it can be.

In transport theory, neutron transport problems are solved by the Boltzmann transport equation. Urban legend claims that after discovering the transport equation, Ludwig Boltzmann killed himself because he could not solve the equation. In reality, the more likely explanation is that Boltzmann battled with severe depression and, at the time, faced significant backlash from the scientific community for his new theories. The combination of depression and his differing ideas from the majority of the scientific community resulted in his sudden death in 1906. In the end, Boltzmann's ideas were vindicated and he is one the most well-known physicists of all time. [11] He is renowned for his significant contributions to the field of statistical mechanics and statistical interpretations of entropy. While Boltzmann's transport equation was a huge contribution to the theory of neutron transport, his death should also serve as an important lesson for us all.

2.2 The Boltzmann Transport Equation

In the following sections, we will do a full breakdown of the BTE. In essence, it is a balance equation, which claims that all neutrons gained in a certain nuclear process are later lost. As a graph it can be expressed as



and as an equation

$$\frac{1}{v(E)}\frac{\partial}{\partial t}\psi(\mathbf{v},\hat{\mathbf{\Omega}},E,t)+\hat{\mathbf{\Omega}}\cdot\nabla\psi(\mathbf{v},\hat{\mathbf{\Omega}},E,t)+\Sigma_{\mathrm{t}}(\mathbf{r},E)\psi(\mathbf{v},\hat{\mathbf{\Omega}},E,t) = q(\mathbf{r},\hat{\mathbf{\Omega}},E,t) \quad (2.1)$$

This is the *time-dependent form of the linear Boltzmann transport equation.* Going forward, we will change this equation for convenience sake and derive a balance equation for the scalar neutron density or *scalar flux* for short. It is determined by integrating the angular neutron flux across the entire space-angle:

$$\phi(\mathbf{r},E) = \int_{4\pi}\psi(\mathbf{r},\hat{\mathbf{\Omega}},E)d\hat{\mathbf{\Omega}} = v\int_{4\pi}n(\mathbf{r},\hat{\mathbf{\Omega}},E)d\hat{\mathbf{\Omega}} = vn(\mathbf{r},E), \quad (2.2)$$

where ϕ is the scalar flux, ψ is the *angluar neutron flux*[1], v is the velocity and $n(\mathbf{r},E)$ is the scalar (or total) neutron density. The result should offer a much more intuitive view concerning particle density.

2.2.1 Time rate of change

The first term of the equation, which mainly deals with the concentration of neutrons over time rate of change

$$\frac{1}{v(E)}\frac{\partial}{\partial t}\psi(\mathbf{r},\hat{\mathbf{\Omega}},E,t) \quad (2.3)$$

can be broken up into two parts. Starting with the left-hand side. The term $\frac{1}{v(E)}$ represents the inverse of the particle's velocity $v(E)$ at energy E. In the context of neutron transport, faster neutrons have lower cross-sections for interactions, so this term helps normalise the equation for different velocities.

The second half, $\frac{\partial}{\partial t}\psi(\mathbf{r},\hat{\mathbf{\Omega}},E,t)$, represents the time rate of change of the neutron distribution function ψ with respect to time t. It describes how the neutron distribution changes over time due to various processes. The function ψ is known as angular neutron flux, which has units $\mathrm{neutrons}/(\mathrm{cm}^2\cdot\mathrm{eV}\cdot\mathrm{s}\cdot\mathrm{sr})$. It is a vector quantity that represents the flow of neutrons per unit area in a particular direction. In other words, it is used to describe the rate at which neutrons are passing through a particular area of space in a particular direction. The quantity $\hat{\mathbf{\Omega}}$, which is measured in two angular variables (η and ν), represents the solid

[1]The angular neutron flux lacks independent physical significance. However, it is related to the rate at which neutrons with energy E stream through space in direction $\hat{\mathbf{\Omega}}$. [3]

angle direction, which is measured in steradians (sr).

$$\begin{aligned}\Omega_x &= \sin(\eta)\cos(\nu)\\ \Omega_y &= \sin(\eta)\sin(\nu)\\ \Omega_z &= \cos(\eta)\end{aligned} \tag{2.4}$$

Integrating this over the total space angle yields

$$\int_{4\pi} \frac{1}{v}\frac{\partial}{\partial t}\psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t)d\hat{\mathbf{\Omega}} = \frac{1}{v}\frac{\partial}{\partial t}\int_{4\pi} \psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t)d\hat{\mathbf{\Omega}} = \frac{1}{v}\frac{\partial}{\partial t}\phi(\mathbf{r}, E, t). \tag{2.5}$$

2.2.2 Leakage term

The second term that deals with leakage

$$\hat{\mathbf{\Omega}} \cdot \nabla\psi(\mathbf{v}, \hat{\mathbf{\Omega}}, E, t) \tag{2.6}$$

is perhaps the most unintuitive compared to the other terms of the equation. This term accounts for the advection of neutrons due to their motion in space. The dot product between the neutron direction vector $\hat{\Omega}$ and the gradient of ψ with respect to position $\nabla\psi$ provides a geometric perspective on the effect of neutrons moving in the medium. Geometrically, the dot product represents the projection of the gradient of ψ onto the direction of the neutron, which shows how the neutron movement influences the spatial distribution of the scalar flux ψ within the medium.

Mathematically, the neutron leakage depends on the net current that flows out of some area dA that encloses a volume. Or in other words

$$\int_A \mathbf{j}(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t)dA. \tag{2.7}$$

The function j is known as the *angular current density* and its measured unit is $\mathrm{neutrons}/(\mathrm{cm}^2 \cdot \mathrm{eV} \cdot \mathrm{s})$. It is a six-dimensional vector quantity that describes the flow of neutrons per unit area in a given direction. In other words

$$\mathbf{j}(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) = \mathbf{v}n(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t), \tag{2.8}$$

or in the terms of angular flux:

$$\mathbf{j}(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) = \hat{\mathbf{\Omega}}\psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t). \tag{2.9}$$

Using these equations, we can write the original term in the new form as

$$\hat{\mathbf{\Omega}} \cdot \nabla \psi(\mathbf{r}, \hat{\Omega}, E, t) = \nabla \cdot \hat{\mathbf{\Omega}} \psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) = \nabla \cdot \mathbf{j}(\mathbf{r}, \hat{\Omega}, E, t) \tag{2.10}$$

Using Gauss's divergence theorem and integrating over the full space-angle, we get

$$\int_{4\pi} \nabla \cdot \mathbf{j}(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) d\hat{\mathbf{\Omega}} = \nabla \cdot \mathbf{J}(\mathbf{r}, E, t), \tag{2.11}$$

where $\mathbf{J}(\mathbf{r}, E, t)$ is the *neutron current density* also measured in units $\mathrm{neutrons}/(\mathrm{cm}^2 \cdot \mathrm{eV} \cdot \mathrm{s})$. Compared to angular current density, neutron current density is a scalar quantity that describes the total flow of neutrons in a certain direction, regardless of the solid angle.

2.2.3 Total collision term

The total collision or the total absorption term describes that every collision will potentially remove a neutron from flux, through absorption or by scattering away from the differential angular and energy space.

This term includes contributions from scattering events where the neutron's direction ($\hat{\mathbf{\Omega}}$) and energy (E) may remain unchanged (i.e., $\hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}'$ and $E \to E'$). To clarify further, if we extract these specific scattering terms and subtract them from the total collision term, we obtain a removal term, signifying neutron removal from the phase space. Consequently, the remaining scattering source becomes a specialized term that only expresses scattering to groups and energies different from the starting ones. In essence, the total collision term comprises both removal and scattering components, emphasizing the intricate interplay between these processes in neutron transport simulations. To conclude the term

$$\Sigma_{\mathrm{t}}(\mathbf{v}, E, t) \psi(\mathbf{v}, \hat{\mathbf{\Omega}}, E, t) \tag{2.12}$$

represents the total macroscopic cross-section for neutron interaction (absorption and scattering) at energy E, velocity $\mathbf{v}$, and time t, multiplied by the neutron distribution function ψ. It accounts for the loss of neutrons due to interactions.

Integration the absorption term gives us

$$\begin{aligned}\int_{4\pi} \Sigma_{\mathrm{t}}(\mathbf{r}, E) \psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) d\hat{\mathbf{\Omega}} = \Sigma_{\mathrm{t}}(\mathbf{r}, E) \int_{4\pi} \psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) d\hat{\mathbf{\Omega}} = \\ = \Sigma_{\mathrm{t}}(\mathbf{r}, E) \phi(\mathbf{r}, E, t).\end{aligned} \tag{2.13}$$

2.2.4 Source terms

The source term can be written out into three separate parts, which are called the external, scattering and fission source. Here is the equation below:

$$q(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) = q_{\text{ex}}(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) + q_{\text{s}}(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) + q_{\text{f}}(\mathbf{r}, E, t) \tag{2.14}$$

The latter two terms, scattering and fission, are dependent on the reaction rates of neutron flux. The reaction rate or the rate-density of a reaction refers to the rate of a particular reaction per unit volume or per unit mass of a substance. It is a measure of the number of reactions that occur in a given volume or mass over a specified period of time. It is described by the equation

$$r_i(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) = \Sigma_i(\mathbf{r}, E, t)\psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) \tag{2.15}$$

where Σ_i is the macroscopic cross-section of the i-th reaction (m^{-1}). However the external source term is independent of flux. It gives the rate at which neutrons are emitted into the phase-space element by external sources.

The likelihood of a neutron with energy E', moving in the direction $\hat{\mathbf{\Omega}}'$, undergoing a scattering event and ending up within a small energy interval dE around E, as well as a differential solid angle $\mathbf{d}\hat{\mathbf{\Omega}}'$ around $\hat{\mathbf{\Omega}}'$, can be characterised through the macroscopic double-differential scattering cross-section. The scattering source is defined by integrating the differential rate of scattering across all possible incident energies and directions:

$$q_{\text{s}}(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) = \int_{4\pi} \int_0^\infty \Sigma_{\text{s}} \left(\mathbf{r}, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}, E' \to E \right) \psi \left(\mathbf{r}, \hat{\mathbf{\Omega}}', E', t \right) d\hat{\mathbf{\Omega}}' dE' \tag{2.16}$$

As fission neutrons are emitted uniformly in all directions, the fission source term remains unaffected by angular variables. Therefore, for convenience, expressing the fission source term in relation to the scalar flux rather than the angular flux is more suitable:

$$q_{\text{f}}(\mathbf{r}, E, t) = \frac{1}{4\pi} \int_0^\infty \chi(E)\nu\Sigma_{\text{f}}\left(\mathbf{r}, E'\right)\phi\left(\mathbf{r}, E', t\right) dE' \tag{2.17}$$

Here, $\chi(E)$ represents the fission spectrum, denoting the probability of an emitted neutron having its energy fall within a small interval dE around E. Basically, the fission spectrum tells us how many neutrons are present at each energy level and it helps us understand how neutrons interact with specific materials. For ease of expression, the term $\nu\Sigma_{\text{f}}(\mathbf{r}, E)$, which denotes the rate of fission neutron production, is treated as a single entity, and the average number of emitted fission neutrons is presented without explicit dependence on energy.

When integrating the scattering source over the full space-angle, we have

$$\int_{4\pi} q_{\mathrm{s}}(\mathbf{r}, \hat{\mathbf{\Omega}}, E, t) d\hat{\Omega} =$$
$$= \int_{4\pi} \left[\int_{4\pi} \int_0^\infty \Sigma_{\mathrm{s}} \left(\mathbf{r}, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}, E' \to E \right) \psi \left(\mathbf{r}, \hat{\mathbf{\Omega}}', E', t \right) d\hat{\mathbf{\Omega}}' dE' \right] d\hat{\Omega} \tag{2.18}$$

However, in order for us to integrate this equation effectively, we would rather prefer that the scattering angle direction order be $(\hat{\Omega} \to \hat{\Omega}')$, which is why we have to make a notable change. As it turns out, the specific order in which we write the directions does not matter, because the scattering cross-section only depends on the angle between the initial and final directions. In other words, whether the reaction is from $(\hat{\Omega}' \to \hat{\Omega})$ or $(\hat{\Omega} \to \hat{\Omega}')$, the scattering effect will ultimately be the same anyway. All of this is possible, because the scattering process does not care about which direction is considered “initial” and which is “final”, as long as the angle between them remains the same. Therefore we can carry on with the integration as such.

$$\begin{aligned}
&\int_{4\pi} \int_0^\infty \left[\int_{4\pi} \Sigma_{\mathbf{s}} \left(\mathbf{r}, \hat{\mathbf{\Omega}} \to \hat{\mathbf{\Omega}}', E' \to E \right) d\hat{\mathbf{\Omega}} \right] \psi \left(\mathbf{r}, \hat{\mathbf{\Omega}}', E', t \right) \, d\hat{\mathbf{\Omega}}' \, dE' \\
&\quad = \int_{4\pi} \int_0^\infty \Sigma_{\mathbf{s}} \left(\mathbf{r}, E' \to E \right) \psi \left(\mathbf{r}, \hat{\mathbf{\Omega}}', E', t \right) \, d\hat{\mathbf{\Omega}}' \, dE' \\
&\quad = \int_0^\infty \Sigma_{\mathbf{s}} \left(\mathbf{r}, E' \to E \right) \left[\int_{4\pi} \psi \left(\mathbf{r}, \hat{\mathbf{\Omega}}', E', t \right) \, d\hat{\mathbf{\Omega}}' \right] \, dE' \\
&\quad = \int_0^\infty \Sigma_{\mathbf{s}} \left(\mathbf{r}, E' \to E \right) \phi \left(\mathbf{r}, E', t \right) \, dE'
\end{aligned} \tag{2.19}$$

2.3 Neutron balance equation

Using all the results we obtained previously and substituting them into the original Boltzmann equation or more exactly the neutron continuity equation, we arrive at a result, which looks like this:

$$\frac{1}{v} \frac{\partial}{\partial t} \phi(\mathbf{r}, E, t) + \nabla \cdot \mathbf{J}(\mathbf{r}, E) + \Sigma_{\mathrm{t}}(\mathbf{r}, E) \phi(\mathbf{r}, E, t)$$
$$= \int_0^\infty \left[\Sigma_{\mathrm{s}} \left(\mathbf{r}, E' \to E \right) \phi \left(\mathbf{r}, E', t \right) + \frac{1}{4\pi} \chi(E) \nu \Sigma_{\mathrm{f}} \left(\mathbf{r}, E' \right) \phi \left(\mathbf{r}, E', t \right) \right] dE' \tag{2.20}$$

This equation is also referred to as the *neutron balance equation.* It is a fundamental equation that describes the behaviour of neutrons in a nuclear system. It accounts for the creation and destruction of neutrons within that system.

To recap, the neutron continuity equation focuses on the flow of neutrons within a nuclear reactor or any other similar system. It describes how neutrons move from one region to another, just like how a river flows from a higher elevation to a lower one. For this it is important to take into account important factors such as leakage, absorption and scattering. In short, the neutron balance equation is concerned with the overall neutron population, while the neutron continuity equation deals with the movement and distribution of neutrons within a system.

2.3.1 The Criticality Eigenvalue Problem

In nuclear systems, such as reactors, the equilibrium state refers to a condition where the number of neutrons produced by fission reactions is balanced by the number of neutrons lost through absorption and leakage. This delicate balance is crucial for maintaining a stable and sustainable nuclear reaction.

The implication of such a balance point implies the existence of a steady-state system. To find it, we turn to the steady-state neutron transport equation. This equation says that there can exist a balance between the fission source rate and the neutron rate of loss caused by absorption and leakage, making the chain reaction stationary and self-sustaining. In its simplest form, which only considers the steady-state conditions, meaning that the neutron flux and other parameters do not change with time and diffusion effects are not considered, looks like this:

$$\hat{\mathbf{\Omega}} \cdot \nabla \psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E) + \Sigma_{\mathrm{t}}(\mathbf{r}, E)\psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E) = q(\mathbf{r}, \hat{\mathbf{\Omega}}, E). \tag{2.21}$$

By solving this equation, we can determine the neutron flux distribution throughout the system and assess whether it reaches a stable equilibrium or not. This is where the concept of criticality is introduced. Now the term itself may cause distress to someone not familiar with the concept. Rest assured, it does not directly mean that a reactor is on the verge of an explosion. Criticality just refers to the condition, where the neutron population in a nuclear system remains constant over time. In other words, it just describes the relation between neutron production due to fission processes and losses that occur due to absorption and leakage.

Criticality is something that can be described by a single scalar value, which is where the concept of the eigenvalue problem comes into play. The k-eigenvalue method involves simulating a stationary self-sustaining chain reaction and finding an effective multiplication factor, denoted as k-effective or just k_{eff} for short. This value describes the ratio of the number of neutrons produced in one generation to the number of neutrons in the previous generation. If this factor is equal to 1, the reactor is in a critical state, which means that the neutron population is stable and self-sustaining. If the factor is greater than 1, the reactor is supercritical, which means that the neutron population is increasing, and if it is less than 1, the reactor is sub-critical, which means that the neutron population is decreasing.

$$k_{\text{eff}} \begin{cases} < 1 & \Longrightarrow \text{ system is sub-critical} \\ = 1 & \Longrightarrow \text{ system is critical} \\ > 1 & \Longrightarrow \text{ system is super-critical} \end{cases} . \tag{2.22}$$

2.3.2 Steady-state equation

To calculate the criticality, we can use the steady-state neutron transport equation along with the concept of the k-effective factor. By adjusting various parameters, such as the geometry, material composition and neutron source strength, we can determine the conditions under which the system reaches criticality.

$$\begin{aligned} &\hat{\mathbf{\Omega}} \cdot \nabla\psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E) + \Sigma_{\text{t}}(\mathbf{r}, E)\psi(\mathbf{r}, \hat{\mathbf{\Omega}}, E) \\ &= \int_{4\pi}\int_0^\infty \Sigma_{\text{s}}\left(\mathbf{r}, \hat{\mathbf{\Omega}}' \to \hat{\mathbf{\Omega}}, E' \to E\right)\psi\left(\mathbf{r}, \hat{\mathbf{\Omega}}', E'\right) d\hat{\mathbf{\Omega}}'\, dE' \quad + \\ &+ \frac{1}{k_{\text{eff}}}\frac{1}{4\pi}\int_0^\infty \chi(E)\nu\Sigma_{\text{f}}\left(\mathbf{r}, E'\right)\phi(\mathbf{r}, E', t)\, dE'. \end{aligned} \tag{2.23}$$

It is worth noting that calculating the equilibrium state and criticality is a complex task that requires sophisticated mathematical techniques and computational methods. There are several ways to find the solution to this equation. Using the stochastic approach, the most common one is the Monte Carlo method. From deterministic methods, which rely on radiative transfer theory, there is the method of discrete ordinates. Both of these methods have their advantages and disadvantages and shall be described in more detail in the following chapters.

2.4 Solution methods

The issue regarding the existence and uniqueness of solutions remains unresolved for the Boltzmann transport equation, but recent findings show promising developments. [12, 13] There are specific cases or simplified scenarios, where analytical solutions are possible, but most often, numerical methods are employed for solving the transport equation in practical applications. The numerical methods can be divided into two main branches: deterministic and stochastic.

A deterministic approach relies on precise predictable rules and conditions. Given the same initial conditions, the outcome of the calculations will always be the same. Therefore, if we assume that our process has clear cause-and-effect relationships, then the deterministic methods are the well-behaved rule-abiding approaches. However, deriving the analytical equations for a deterministic approach can be mathematically challenging and may require a high level of expertise to implement. Then again, once you have the analytic solution, it does not usually require heavy computations and is therefore time and resource conservative.

On the other hand, when modelling the chaotic movement of small particles, such as neutrons, a stochastic approach allows us to embrace randomness and uncertainty into the equations. Stochastic approaches, such as Monte Carlo, more closely capture the statistical nature of neutron transport. They are also much easier to implement given the simplistic nature of Monte Carlo. However, simulating a large amount of individual neutrons in a reactor physics simulation can prove to be quite computationally heavy to run. If you do not know any good optimization methods or have access to a computer cluster to run your computations on, it may be better to settle for a more deterministic approach.

While the choice between a deterministic or a stochastic approach hinges on the individual needs and hardware availability of the user, the decision becomes nuanced when dealing with substantial nuclear reactors. Attempting to conduct a deterministic neutron transport simulation for an entire large reactor on a standard desktop computer is often impractical, primarily due to memory limitations. This is due to the fact that storing cross-sections in numerous energy groups and directions becomes computationally very burdensome[1].

It is important to note that the Monte Carlo method has its own limitations, particularly when dealing with safety and analyses that require transient simulations. In theory, it is possible to run Monte Carlo simulations in reverse, but in practice it proves to be extremely difficult. Therefore there is no time derivative term in the Monte Carlo method. Despite these challenges, the two-step methodology offers a viable compromise, allowing for efficient memory and computational resource utilisation in deterministic simulations while addressing the shortcomings of the Monte Carlo method in certain analyses.

[1]The Monte Carlo method, which is known for being inherently computationally extensive, paradoxically emerges as a more suitable option for extensive reactor simulations. However, executing a comprehensive Monte Carlo simulation for an entire reactor core in one go can be excessively demanding. To address this challenge, a practical solution involves breaking down the simulation into two stages. [14]

In the first stage, a simulation is run to derive condensed and homogenised cross-sections for specific regions of the reactor. This involves employing the method of collision probability to condense energy information and the Method of characteristics to homogenise it. This is commonly referred to as the "two-step methodology". The aim of this approach is to combine memory management and the available computational power, in order to make deterministic simulations feasible on a standard laptop or desktop computer. [15]

2.4.1 The Monte Carlo Method

From stochastic methods, Monte Carlo is the go-to method for modelling nuclear reactions and interactions with other types of matter. The effectiveness of the Monte Carlo method lies in its ability to compute statistical estimates for integral reaction rates without the need to explicitly solve for the flux distribution. Its capacity to handle intricate variations in spatial and energy variables makes Monte Carlo calculations an appealing alternative to deterministic transport methods. [3]

While the introduction of randomness into the process of modeling neutron transport is a distinctive characteristic of Monte Carlo methods, it is important to acknowledge that randomness comes with inherent challenges. Unlike deterministic methods, which provide a certain outcome every time, randomness introduces complexities such as difficulties in assessing convergence. Determining the appropriate number of neutrons and histories for large systems becomes a challenging task. Even metrics like the Shannon Entropy[1] offer only approximate insights rather than guarantees.

In the context of neutron transport in a two-dimensional unit cell, such as the one found in a pressurised water reactor, the Monte Carlo method can be used to simulate the behaviour of individual neutrons as they move through the reactor core. The main idea behind this method is to simulate a large number of individual neutron histories, track their positions and interactions as they traverse the unit cell. For interactions specifically, a method called Woodcock delta tracking is employed. The specifics of this method will be explained in greater detail below.

The algorithm of the Monte Carlo method can be summed up with these main steps. At the start of the code, neutrons are generated and randomly spread out in the unit cell. All neutrons are described by their location, weight and energy number. The location is a simple 2D coordinate used to keep track of the path each neutron takes. The weight is essentially a percentage value, explaining the "importance" of each individual neutron and the energy number is an index used to keep track of the energy level of each neutron. After all of this is done, we begin the main iteration cycle, at the end of which it is possible for us to calculate the k-effective value. In Monte Carlo simulations it is defined as the multiplication factor in cycle n:

$$k_n = \frac{\text{number of source neutrons in cycle } n+1}{\text{number of simulated histories in cycle } n}. \tag{2.24}$$

[1]Shannon entropy, named after Claude Shannon, is a measure of uncertainty or information content in a set of possible outcomes. A higher entropy in this context could indicate higher uncertainty and the need for additional simulation runs to achieve convergence. [16]

The main iteration cycle loops over all the neutrons, which undergo a random walk cycle from emission to absorption. These neutron cycles or, in other words, generations, are run hundreds of times to accumulate the data to calculate the reactor's criticality value. For this, several factors need to be monitored to ensure a realistic reaction process. To control the number of neutrons in the system, an implicit Monte Carlo simulation strategy is employed during each cycle. In this approach, a form of *Russian roulette* is played, randomly eliminating neutrons that might undergo potential absorption reactions. Unlike the analog Monte Carlo simulations, where neutrons are either born or die without the use of weights, the implicit method introduces the concept of particle weights as a clever strategy to enhance simulation efficiency. A straightforward approach involves expressing the termination probability like this:

$$P = 1 - \frac{W}{W_0} \tag{2.25}$$

where W_0 represents the neutron weight at the start of a neutron generation. If the neutron successfully passes the test, its weight is reset to W. This helps maintain a realistic neutron population and eliminates the excess number of neutrons that can be produced during an exponential neutron growth event, such as multiple consecutive fission reactions. [3]

During the simulation, a neutron can undergo an inelastic scattering event, where it can be absorbed by the target nucleus. If a neutron gets absorbed or does not contribute to any meaningful reaction, it is important to clean up these absorbed or unnecessary neutrons in order to conduct proper memory management. However, fissionable nuclei may be split as well, which introduces new neutrons into the system. Therefore careful measures are taken to ensure the existence of sufficient number of neutrons within the system.

All of this and more will be explained in more detail in the methodology section, along with detailed descriptions with examples from the code. On paper, the Monte Carlo may seem to be deceptively simple approach for simulating the k-effective value, but the underlying theory behind this method is quite complex and contains certain nuances that require great attention to detail for simulating the process realistically.

The Woodcock delta-tracking method

Traditional Monte Carlo particle transport methods typically rely on ray-tracing algorithms, demanding the use of complex geometry routines. An alternative approach to the conventional ray-tracing algorithm is the delta-tracking method, introduced by E. R. Woodcock in 1965. [17] Unlike the conventional method, delta-tracking samples the next collision point without dealing with surface crossings. In instances where a neutron undergoes an interaction without absorption, preserving both incident energy and flight direction, it is termed a *virtual collision*, denoted by the cross-section $\Sigma_0(r, E)$. [18]

The virtual collision cross-section, $\Sigma_0(\mathbf{r}, E)$, is defined as

$$\Sigma_0(\mathbf{r}, E) = \Sigma_\mathrm{m}(E) - \Sigma_\mathrm{t}(\mathbf{r}, E), \tag{2.26}$$

where $\Sigma_\mathrm{t}(\mathbf{r}, E)$ is the physical total cross-section of the material and $\Sigma_\mathrm{m}(E)$ is the maximum total cross-section of the material, known as the majorant. The majorant can be defined as:

$$\Sigma_\mathrm{m}(E) = \max[\Sigma_\mathrm{t}(\mathbf{r}, E)]. \tag{2.27}$$

Overall, the delta-tracking method is a type of rejection sampling technique. These methods are frequently employed when sampling values from probability distributions where computing quantile functions is computationally expensive or cannot be easily solved. In the delta-tracking method, the aim is to homogenize the total cross-sections of the material effectively, ensuring that the sampled path lengths[2] remain applicable throughout the entire geometry. [18]

In a traditional ray-tracing method, the flight path is halted at the boundary surface. A new path length is then sampled based on the updated material total cross-section. In practice, the geometry routine calculates the distance to the nearest boundary surface and compares it to the sampled path length. In the Monte Carlo simulation's random walk, the lengths of neutron paths are obtained by sampling from an exponential distribution

$$l = -\frac{1}{\Sigma_\mathrm{m}} \log \xi, \tag{2.28}$$

where ξ is a uniformly distributed random variable on the unit interval. The path lengths obtained with the global majorant are statistically valid for all materials. There's no requirement to pause the random walk at boundary crossings, eliminating the need for calculating surface distances. Instead, a tracking routine introduces an extra step to handle virtual collisions. [18] Rejection sampling is applied to each collision, and the collision point is accepted with a certain probability, which can be caluclated as

$$P = \frac{\Sigma_\mathrm{t}(\mathbf{r}, E)}{\Sigma_\mathrm{m}(E)} \tag{2.29}$$

The Woodcock tracking method has been incorporated into numerous nuclear codes, serving diverse applications such as shielding [19], medical [20], reactor simulations [18] as well as computer graphics [21, 22]. Its primary advantage lies in simplifying geometry routines and potentially accelerating calculations in intricate geometries. A

[2]Also known as the mean free path. It is the distance a neutron travels between two collisions or interactions.

notable drawback is the method's failure to record surface crossings, resulting in the absence of a track-length estimate for neutron flux. An even more significant challenge associated with delta-tracking is encountered when there is a localized absorber in a specific part of the geometry. This leads to a major drawback, where the tracking becomes strongly tied to that absorber region, resulting in an inefficient sampling of paths throughout the entire geometry. [18]

2.4.2 Method of Discrete Ordinates

The second method that we will be using for the deterministic approach is called the method of discrete ordinates or the S_N method. Originally developed by Chandrasekhar in 1960, it is a versatile and widely used numerical method in nuclear engineering and radiation physics. [23, 24]

The neutron transport equation governs the transport of neutrons in the system. For reactor design and shielding issues, a particularly valuable parameter is the angular flux ψ. This flux represents the product of neutron density at a specific point in the angular phase space and velocity of neutrons. Compared to the scalar flux ϕ, the angular flux has the advantage of retaining the crucial information about the direction in which the neutrons are moving.

One crucial aspect about the discrete ordinates method is the approximation of angular integration. Calculating such surface integrals becomes very important, when expressing the right-hand side components of the Eq.(2.23). Neutrons can travel in various directions, which is why the main theory behind using this method is to model a finite set of possible angles (i.e. *discrete ordinates*) a neutron can take in order to discretize the angular variables of neutron angular flux. [25, 26]

Lebedev-Laikov quadrature

It is important to consider that calculating such surface integrals needs to be done often and with a high degree of accuracy. An effective approach to address this problem is to employ the Lebedev-Laikov quadrature method for solving definite surface integrals. While widely used, it is worth noting that other methods, such as Gauss quadratures, may also be common in this context, though the prevalence may vary depending on specific applications and preferences. Lebedev-Laikov quadratures excel in providing highly accurate and efficient rules for approximating integrals of the following form:

$$\int_S f(\mathbf{r})d\mathbf{r} = \int_S x^2 + y^2 - z^2 \, dx \, dy \, dz. \tag{2.30}$$

The parametric representation for the position variables can be expressed as

$$x = r\ \sin(\eta)\cos(\nu) \tag{2.31}$$

$$y = r\ \sin(\eta)\sin(\nu) \tag{2.32}$$

$$z = r\ \cos(\eta) \tag{2.33}$$

where r is the radius of the sphere, ν is the elevation angle and η is the direction angle. Additionally, the limits of the angles are defined as $\nu \in [0, \pi]$ and $\eta \in [0, 2\pi]$. The discretized volume element can be expressed by two ways: the determinant of the Jacobian or the cross product of tangent vectors. The former is formulated like this:

$$|\mathrm{J}| = \begin{vmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \nu} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \nu} & \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial r} & \frac{\partial z}{\partial \nu} & \frac{\partial z}{\partial \eta} \end{vmatrix} = \begin{vmatrix} \sin(\eta)\cos(\nu) & r\cos(\nu)\cos(\eta) & -r\sin(\nu)\sin(\eta) \\ \sin(\nu)\sin(\eta) & r\sin(\nu)\cos(\eta) & r\sin(\eta)\cos(\nu) \\ \cos(\eta) & -r\sin(\eta) & 0 \end{vmatrix} = \\ = r^2\ \sin(\eta). \tag{2.34}$$

Our initial function can now be written in the form $f(r, \hat{\mathbf{\Omega}})$. As we can see, it depends on the radius of the sphere and the two angles η and ν. Therefore the surface integral (2.30) can be expressed as

$$\int_S f(r, \eta, \nu)|\mathrm{J}|\, d\eta\, d\nu, \quad 0 < \eta < \pi,\ 0 < \nu < 2\pi. \tag{2.35}$$

Using the latter approach to calculate the volume element we get

$$T_\eta \times T_\nu = \begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ \frac{\partial x}{\partial \nu} & \frac{\partial y}{\partial \nu} & \frac{\partial z}{\partial \nu} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta}, \end{bmatrix} \tag{2.36}$$

where the volume element is also

$$|T_\eta \times T_\nu| = r^2\ \sin(\eta). \tag{2.37}$$

This time the final form for the Eq. (2.30) can be expressed as

$$\int_S f(r, \eta, \nu)|T_\nu \times T_\eta|\, d\eta\, d\nu, \quad 0 < \eta < \pi,\ 0 < \nu < 2\pi \tag{2.38}$$

Finally, the full form of the Eq.(2.39) in spherical coordinates can be written as

$$\int_0^{2\pi}\int_0^{\pi}\left\{\left[(r\,\sin(\eta)\cos(\nu))^2+(r\,\sin(\eta)\sin(\nu))^2-(r\,\cos(\eta))^2\right]\cdot r^2\,\sin(\nu)\right\}d\eta d\nu \tag{2.39}$$

Since we are calculating integrals for a unit sphere, we can set $r = 1$. Therefore our input function only depends on the angles η and ν, which can be represented as the solid state angle $\hat{\mathbf{\Omega}}$. This means that our integral only depends on the solid angle, i.e.

$$\int_{S'} f(\mathbf{r})d\mathbf{r} = \int_S f(\hat{\mathbf{\Omega}})d\hat{\mathbf{\Omega}}. \tag{2.40}$$

It is important to note that solving surface integrals in this context is quite similar to how one-dimensional integrals are solved using the Gauss-Legendre quadrature rules. It involves approximating an integral with a discretized sum, using the roots and weights of Legendre polynomials, which are used to calculate the neutron flux integrals in various directions. The Lebedev-Laikov quadrature uses the spherical harmonics $T_{lm}(\eta,\nu)$ as basis functions. The associated Legendre polynomials $P_l^m(cos(\eta))$ are the angular part of these spherical harmonics and the roots of the associated Legendre polynomials are employed to determine the angular nodes on the unit sphere. [27] To evaluate the Eq.(2.39) using the Lebedev quadrature rules, we arrive at our approximation scheme, which can be written as

$$\int_S f(\hat{\mathbf{\Omega}})\,d\hat{\mathbf{\Omega}} \approx \sum_{i=1}^{N} f(\hat{\mathbf{\Omega}}_i)\cdot w_i \tag{2.41}$$

Herein lies the main principle, which describes, what the S_N approximation method stands for. Essentially, the total solid angle $\hat{\mathbf{\Omega}}$, which is equal to 4π, is discretized into N sectors with surfaces w_i, where $i = 1, 2, 3, ..., N$. These surfaces are associated with N directional bins, or in other words, discrete ordinates $\hat{\mathbf{\Omega}}_i$, in order to numerically calculate the angular flux and other derived quantities, such as the scalar flux and current, power distribution and so on. Furthermore, the greater the value for N is, the more computationally expensive the calculations become and the task we are solving is not the easiest. [26, 27]

However, in numerical applications, the total solid angle $\hat{\mathbf{\Omega}}_i$ is substituted for the directional cosine μ_i, which is defined as the cosine of the angle between the particle's direction and a reference axis. The substitution simplifies the mathematical representation of the transport equation, providing a more convenient way for doing numerical analysis. For example, if μ_i is 1, the particle is moving parallel to the reference axis; if μ_i is -1, the particle is moving in the opposite direction, and if μ_i is 0, the particle is moving perpendicular to the reference axis. Additionally, since the angular flux is represented by a discrete set of directions, we have the opportunity to define whether the scattering source is isotropic or anisotropic.

Solver for System of Equations

Since the the steady state Boltzmann transport equation is a linear differential equation, means that it can be written in the form of $Ax = b$. In this equation A is the linear operator, which accounts for the left-hand side of Eq. (2.23) x is the angular flux and b is the source term, i.e., the right-hand side of Eq. (2.23). There are many different methods and approaches to choose from for solving this system of linear equations. Our main conditions for our potential solver are the following. The system of linear equations is most likely non-symmetric and very large. Therefore the solver must be able to compute large matrices fast and efficiently. Secondly, the cross-sectional data can be very complicated, which is why the solver also needs to be quite robust, meaning it has a small sensitivity to initial conditions, and needs to exhibit good convergence properties for a wide variety of matrices. A solver for our choice and which we will be using going forward is called the BIConjugate Gradient STABilised iteration method or BiCGSTAB for short. [28]

In summary, the discrete ordinates method uses the Lebedev quadrature rules to approximate the angular integration for the Boltzmann transport equation. This generally boils down to solving a large system of linear equations every iterative cycle. DOM is a powerful technique that helps understand and analyse the behaviour of neutrons in a nuclear reactor.

3. Methodology

In the following chapters, we will be looking at the step-by-step process of preparing the nuclear cross-section data for calculations and showing two separate methods of solving the Boltzmann transport equation. Our main software for achieving these tasks is meant to be understandable for beginners, easy to use and versatile enough for our problem. Hence why we will be using Python to show how each step of the process is done.

3.1 GENDF Files

Our first step involves downloading the Generalised Evaluated Nuclear Data File from the International Atomic Energy Agency web-page. The GENDF is a type of nuclear data file that was developed by the US around the mid-20th century.

During that time, one of the most popular programming languages used for scientific computing was the Fortran programming language. These were the early days of programming and it was common for Fortran developers to use punch cards to execute code. These punch cards could physically hold a maximum of 80 characters on a line, so a standard was developed of how the nuclear cross section data would be written. The system for organising how the data was written on those punch cards laid the groundwork on how these data files are shaped today.

In short, GENDF-format libraries are computer-readable files of nuclear data that contain data on nuclear cross sections, energy distributions, reaction products, various nuclei decay and so on. This data is used for a wide variety of applications that require calculations for neutron transport, charged-particle interactions with matter and time-dependent evolution of radioactive processes. Examples include fission and fusion reactions, shielding and radiation protection calculations, nuclear weapons, medical radiotherapy, accelerator design and many other fields including geological and environmental work. Our main goal of using these files is to calculate the criticality of a typical pressurised water reactor.

As described in the previous chapter, to model the criticality of a simple nuclear reactor, we need three things: fuel, coolant and structural material. For fuel, the most common option is to use enriched uranium. Natural uranium contains around 0.7% of ^{235}U that fissions and produces energy. The remaining 99.3% is mostly ^{238}U and while its contribution to the fission process is not as significant as ^{235}U, it is noteworthy. Up to 5%-10% of fast fissions can be attributed to ^{238}U. This is evident in the neutron spectrum, where a substantial portion is in the fast range and the non-zero fission cross-section in this range indicates the occurrence of fast fission. Since the quantity of ^{235}U is so low in natural uranium, it is enriched from 0.7% to 3%-5%, which is the normal amount for a low-enriched uranium in a PWR. So the main fuel isotope files that we need to download are ^{235}U and ^{238}U. [29]

For coolant, the most commonly used material is boron water. Natural boron primarily consists of two stable isotopes: ^{10}B and ^{11}B. The primary neutron absorber ^{10}B makes up around 19.9% and the rest, 80.1%, is made up of ^{11}B. The former has a high probability of capturing thermal neutrons which help control the reactor's neutron population. The latter however has a very small cross section and is very ineffective as a neutron absorber. For water the only isotopes we need are ^{1}H and ^{16}O. [30]

As a structural element, Zirconium inherits excellent neutron transparency and corrosion resistance properties. For nuclear applications, the most relevant isotopes are ^{90}Zr, ^{91}Zr, ^{92}Zr, ^{94}Zr and ^{96}Zr. [31]

3.2 Data preparation and preprocessing

The GENDF-format has a hierarchical structure by tape, material, file and section. For separation between these layers, a specific numerical identifier has been assigned to each layer.

To begin with, the term "tape" refers to a formatted data file that contains a collection of nuclear data for a specific material and for a range of energies. Nowadays, the data is not stored on magnetic tapes anymore, but the term "tape" has remained due to historical reasons. The MAT label refers to the GENDF material. The specific number is calculated based on the atomic (Z) and mass number (A). Some examples include 125 for ^{1}H, 525 for ^{10}B and 9228 for ^{235}U. After this we have the MF number, which specifies the type of data. [32] Some examples include:

- MF=1 contains descriptive and miscellaneous data,
- MF=2 contains resonance parameter data,
- MF=3 contains reaction cross sections vs energy,
- MF=4 contains angular distributions,
- MF=5 contains energy distributions,
- MF=6 contains energy-angle distributions,
- MF=7 contains thermal scattering data,
- MF=8 contains radioactivity data

- MF=9-10 contain nuclide production data,
- MF=12-15 contain photon production data,
- MF=30-36 contain covariance data.

Finally the MT number labels the GENDF section. By sections we mean the data for different reactions that might occur. For example, MT=2 stands for elastic scattering or a $(\mathrm{n}, \mathrm{n}_0)$ reaction. MT=18 stands for a fission reaction, meaning $(\mathrm{n}, \mathrm{fission})$ and MT=102 stands for radiative capture or a (n, γ) reaction. Furthermore, each reaction also has distinct energy values at specific temperatures as well.

There are many of these reactions, but the specific MF and MT numbers that we need for given temperatures will be specified later. First we need to make our raw GENDF data more malleable for further development.

3.3 GENDF-to-HDF5 pipeline

To start off, let us look at a specific GENDF data file and see how we have to shape it and process it to make it easier to work with. For this let us look at how the raw data looks like when it is downloaded from the IAEA website. [33]

To illustrate our example, this is how the ^{235}U file looks like. The data is specified with a distinct extension named ".GXS", which stands for Generalised (or Group) XS (Cross Sections). Here is what the first six lines of that look like:

U_235.GXS

```
9.223500+4 0.000000+0          1         10          0       4219228 3102    1
2.936000+2 0.000000+0          2          1         20         19228 3102    2
2.407191-7 2.338599-7 2.192808-7 1.861296-7 1.217258-7 6.121356-89228 3102    3
4.088789-8 2.234181-8 7.938595-9 8.18150-10 2.469122+3 2.469122+39228 3102    4
2.469123+3 2.469122+3 2.469123+3 2.469122+3 2.469122+3 2.469122+39228 3102    5
2.469122+3 2.469122+3                                          9228 3102    6
```

At the current stage this nuclear data is quite hard to read. Notice how the scientific notation for numbers does not contain the exponent "E". Again, this was done so to conserve the physical space on punch cards. Also some numbers are written up to five comma places, some six. Furthermore, because it was so important to conserve the space on punch cards, there is no separation index added to these numbers, which results in different columns of data being written right next to each other.

So our first step is to convert this GXS data into a more familiar format that is easier to read and work with: comma-separated values (CSV). This means we have to iterate through each GXS file that we have and then, line-by-line, manipulate and separate the data. A code that implements this task is named `convertGXS2CSV.py` and can be found in the included GitHub repository (in Appendix 4) inside the folder `01.Micro.XS.421g/`. After processing all the nuclear data, this is what we are left with:

U_235.CSV

```
9.223500E+4; 0.000000E+0;           1;          10;           0;         421;9228; 3;102;    1
2.936000E+2; 0.000000E+0;           2;           1;          20;           1;9228; 3;102;    2
2.407191E-7; 2.338599E-7; 2.192808E-7; 1.861296E-7; 1.217258E-7; 6.121356E-8;9228; 3;102;    3
4.088789E-8; 2.234181E-8; 7.938595E-9; 8.18150E-10; 2.469122E+3; 2.469122E+3;9228; 3;102;    4
2.469123E+3; 2.469122E+3; 2.469123E+3; 2.469122E+3; 2.469122E+3; 2.469122E+3;9228; 3;102;    5
2.469122E+3; 2.469122E+3;            ;            ;            ;            ;9228; 3;102;    6
```

As you can see, the data is much more readable thanks to the separating characters and properly formatted numbers. The first value at the first line is the reaction Q-value[1] (9.223500E+4 MeV) and on the same line we have the integer value 421, which specifies the number of groups that energy is distributed into. At the start of the second line we have the accoring reaction temperature (293.6 K) and on the same line the number 20 specifies the number of interpolated data points that begin from the next line.

To illustrate the hierarchical labels that we introduced before, we can read out some important numbers on the right side columns. Firstly, 9228 is the MAT number which corresponds to the element ^{235}U. The adjacent column with number 3 is the MF number, which means this section describes the reaction cross sections in contrast to energy data. And finally, 102 is the MT value, which represents the radiative capture reaction. The main body of this section contains the 20 interpolated data points which contain the nuclear cross-sectional data. [32, 34, 35]

3.4 Formatting microscopic cross sections

Now that our nuclear data has been converted from .GXS to .CSV for easier handling and processing, we can start organising our data for each isotope file. Since we are writing in

[1]The energy released or absorbed in a nuclear reaction.

Python, we should also put some effort into code speed optimization, because this step can take a lot of time if not properly handled.

Starting off, we need to go through each CSV file and separate each isotope's data based on the temperature values. A simple way to find how many different temperature groups our original data includes and what they are, is to use the values "1; 451; 2" for the last three column values. MF=1 stands for general information, MT=451 for comments and dictionary and 2 is the line number at current subsection. At the start of each section, that has these three numbers at the end, is a temperature value that we will be using to organise our data. Since the cross-sectional data files contain large amounts of data and must be organised in a coherent way, we will be using the Hierarchical Data Format (HDF or HDF5) to create a dictionary file for every isotope.

Storing our nuclear cross-section data in a HDF file offers several advantages. First of all, it is very fast at displaying large and complex datasets. Secondly, it is a well-established file format with a large user community and extensive documentation. Plus it is quite simple to use in Python. Thirdly, it allows for a wide range of data types: integers, floating-point numbers, text, images and more. Fourthly, it is platform independent meaning data can be created, read and manipulated on different operating systems and with different programming languages. Finally, it also offers metadata support. This information helps describe and annotate the data, making it easier to understand and use in the future. Naturally, these are not the only benefits of working with HDF files, but these are the most important for us.

To start with our data organisation, we first write some metadata about every isotope that we will be using later from the CSV files. It includes information such as the atomic weight, the number of energy groups, temperature and the number of sigma-zeros. After that we start extracting specific reactions from MF=3 and MF=6. Here is a list all the MT reactions we need:

- MT=2: (n,n0) - elastic scattering
- MT=16: (n, 2n) - production of two neutrons
- MT=18: (n,fission) - total fission (sum over MTs 19-21 and 38)
- MT=51-92: (n,ni) - inelastic scattering to excited states; production of a neutron leaving the residual nucleus in the i-th excited state (i = 1, 2,...,40)
- MT=102: (n, γ) - neutron capture reaction
- MT=107: (n, α) - Production of an alpha particle
- MT=221: (N,2N3P)+(N,X) - free gas thermal scattering
- MT=222: (N,3N3P)+(N,X) - thermal scattering for hydrogen binded in water

- MT=278: (N,3N5P)+(N,X) - chi distribution
- MT=452: nubar - average number of neutrons produced per fission even

3.5 Constructing macroscopic cross-sections

After we have organised our microscopic cross sections, it is time to start forming cross-section for materials, pure or compounds. This means we have to start another step of organising our microscopic data into macroscopic cross-sections, since it does not make sense to look at individual neutron-nucleus reactions since atoms are always bound to one another in the material structure. For this we need to generate the data for three main macroscopic materials: fuel, cladding and coolant.

Each macroscopic cross-section has some unique tweaks to finalise the code, but these are the main steps that are common in all of the codes. Firstly, all three files check for the existence of the HDF5 dictionary file named `initPWR_like.h5`. If it does not exist, it is created by a specific function once, meaning later if other macroscopic files are run, there is no need to create the dictionary again. This Python function creates and initialises a hierarchical data structure stored in the HDF5 file that represents a simplified PWR-like nuclear assembly model. The data structure contains three main groups: “g”, “th” and “fr”, which store information about geometry, thermal parameters and fuel rod parameters respectively.

Secondly, all the microscopic cross-sections for a specific temperature are read in. The user can choose the temperature values, which need to be imported to run the reactor under desired temperature conditions. After that, the third important step is to get the material properties for fuel and cladding. Again, similar to how `initPWR_like.h5` dictionary was created, another file named `matpro.h5` is created to store the data on different material properties for UO_2 (uranium dioxide) fuel, Zircaloy cladding and the gas gap between them. These material properties include density, specific heat, thermal conductivity, thermal expansion, Young’s modulus, Poisson’s ratio, swelling rate, thermal creep rate and additional auxiliary functions. All this data is provided as constant values or as formulas representing temperature-dependent properties.

However, for coolant, i.e. boron water, the approach needs to be a bit different and the necessary data is not retrieved from the material properties file. As it was mentioned before, the user can specify the isotope temperatures for different reactor simulations. Seeing as coolant is mostly made up of water, we need to know properties for steam and water under specific density, temperature and pressure conditions. The `pyXSteam` Python package offers a convenient interface for performing such calculations. It was originally released for MATLAB and Excel and is based on the widely accepted and authoritative international standard, The International Association for the Properties of Water and Steam (IAPWS) Release IF-97. It provides a comprehensive and accurate set of equations for calculating the thermodynamic properties of water and steam. [36]

The fourth step is of utmost importance in creating the macroscopic cross-sections. It is the function that calculates the background or dilution cross-section, named `sigmaZeros()`. It utilises an iterative process to compute the sigma-zero values, which is a measure of how much other nuclei in a material affect the behavior of a specific type of nucleus. It helps us understand the proportion of this specific type of nucleus in the material and how it influences the cross-sections of neutrons. In reality, this means that the process iterates until there is only one isotope present in the mixture given some tolerance value.

To elaborate further, self-shielding involves the idea that one nuclei can influence the effective cross-section of another nuclei of the same type. This concept is particularly significant because it is challenging to describe how easily a neutron can interact with these nuclei. Imagine you have two types of nuclei, i and j, in a material. If j nuclei are like an obstacle to neutrons, it becomes difficult for neutrons to interact with i nuclei. This means the proportion of i nuclei in the material is low and the self-shielding effect is small. On the other hand, if j nuclei allow neutrons to easily reach i nuclei, the proportion of i nuclei is high, leading to a strong self-shielding effect.

Therefore sigma-zero indirectly tells us about self-shielding by helping us figure out how much the presence of other nuclei affects the behavior of a specific type of nucleus. This is why we calculate sigma-zero for different situations – to understand the impact of self-shielding and obtain accurate interaction probabilities for neutrons. The formula which tabulates this multi-group cross-section data is known as the self-shielding equation for background cross-sections:

$$\sigma_{0g}^{i} = \frac{1}{N_i} \sum_{j \neq i} N_j \sigma_{tg}^{j} \left(\sigma_{0g}^{j}, T\right) \tag{3.1}$$

It calculates the background cross-section (σ_{0g}^{i}) for a specific isotope (i) in a particular energy group (g). Let us break down each component in this equation:

- σ_{0g}^{i}: Background cross-section of isotope i in energy group g.
- N_i: Number density of isotope i.
- σ_{tg}^{j}: Total cross-section of isotope j in energy group g.
- N_j: Number density of isotope j.
- σ_{0g}^{j}: Background cross-section of isotope j in energy group g.

- T: Temperature.

Essentially, this equation sums up the product of the total cross-section of each isotope (j) and its background cross-section, weighted by the number density of that isotope. This accounts for the contribution of non-fissile isotopes to the total macroscopic cross-section.

In code, this equation is implemented in the loop over energy groups (`ig`). Specifically, it is part of the inner loop over isotopes (`nIso`) in this section:

```
for iIso in range(nIso):
    # ...
    for jIso in range(nIso):
        if jIso != iIso:
            summation += sigT[jIso, ig] * aDen[jIso]
```

Here, `sigT[jIso, ig]` corresponds to σ_{tg}^{j} and `aDen[jIso]` corresponds to N_j. The summation variable accumulates the product of these values. Then, this accumulated value is used in the calculation of `tmp`:

```
tmp = (SigEscape + summation) / aDen[iIso]
```

In code, an additional term named `SigEscape` (i.e. escape cross-section Σ_{Escape}) is also part of this equation. This accounts for neutrons that escape from a material or system without undergoing any interactions. For simple convex objects (such as plates, spheres, or cylinders) is given by $S/(4V)$, where V and S are the volume and surface area of the object, respectively. Such information about object shapes can be read from the `initPWR_like.h5` file. This value is then used in the error calculation and updating of the background cross-section:

```
err += (1 - tmp / sig0[iIso, ig])**2
sig0[iIso, ig] = tmp
```

This process iterates until the error falls below a specified tolerance, which is 10^{-6} in this case. After that is done, it is necessary to perform interpolation of microscopic cross-sections for capture, scattering and fission events based on the found sigma-zero values.

Fifth important step is to apply a function named `interpSigS()` to find scattering matrices for sigma-zero values. We need to ensure that the correct cross-sections are used for neutrons of different energies and isotopes. In essence, this function serves as a correction factor to obtain accurate scattering cross-sections for further computations that will be done using Monte Carlo and discrete ordinates.

The final steps include some finishing touches, such as converting cross-sections into correct units, rounding the values up to the same accuracy as the original GENDF files had and writing the final macroscopic data into appropriate HDF5 files. All of these steps are necessary to create cross-sections for fuel, cladding and coolant.

3.6 The Monte Carlo simulation

The first method of approximating the criticality value implements the Monte Carlo approach. The basic idea behind the Monte Carlo method is to simulate a large number of individual neutron histories, tracking their positions and interactions as they traverse the unit cell. The collection of events is based on discrete events, such as scattering or absorption, which is used to assess the final criticality value that evolves over the multiple cycles that the simulation runs over.

As a clarification, it is important to note that our code technically does not *solve* the steady-state Boltzmann transport equation for neutron transport in the conventional sense. As mentioned earlier, the approach involves calculating integral reaction rates by considering the stochastic movements of neutrons without explicitly solving for the flux distribution. In essence, while not directly solving the equation, the code addresses an equivalent problem to obtain the necessary results.

At the start of the code, the user has to input the initial number of source neutrons to start off the reaction. After that it asks for the number of active and inactive cycles for k-effective accumulation. This is one of the crucial characteristics of the Monte Carlo method. The number of inactive cycles, specified by the variable `numCycles_inactive`, determines the number of initial cycles to be skipped before starting the accumulation of the k-effective value. These inactive cycles allow the neutron populations to reach an equilibrium state, which means that during these cycles the neutron populations evolve, but the k-effective value is not recorded or considered. After the number of inactive cycles is defined, we input the number of active cycles for the active phase, where the k-effective value is calculated and accumulated. It is essential to note that, technically, the accuracy reaches a plateau after an extensive number of cycles. Increasing the number of active cycles helps reduce uncertainty to a minimum determined by stochastic noise, influenced by factors like the number of sampled neutrons and the proximity to the actual neutron source state.

The simulation starts with inactive cycles because the initial neutron source distribution may not accurately represent the real solution, possibly being uniform, for instance. Inactive cycles are utilized to approach the actual neutron source. From this point onward, each active cycle contributes to the precision (distinct from accuracy) of the simulation. The number of neutrons in each cycle, coupled with the neutron distribution used, determines the precision achievable for that specific cycle. As the simulation progresses, the neutron source distribution is continually updated, bringing the simulation closer to the actual source. Running a sufficient number of cycles allows the simulation to converge toward the real source distribution, gradually improving accuracy up to the limit defined by the precision offered by the neutron sampling size.

After that the final values the user has to input is the pitch size and the dimensions for the fuel and coolant regions. The pitch represents the size of the unit cell in a square lattice, meaning it is the distance between two fuel pins and influences the spatial distribution of neutron flux and reaction rates within the reactor core. The fuel and coolant region values define the spatial extents of the fuel and coolant regions within the unit cell. Properly specifying these regions is essential for obtaining realistic simulation results.

As we start to simulate the movement of neutrons through the system, it is important to keep track of a few important values. Since we need to keep track of every neutron's location within the system, each neutron is also assigned a statistical weight, which basically determines the importance of the particle. Furthermore, each neutron is also described by the energy group value to determine how neutrons interact with materials in the reactor. We also keep track of all the scattering events as well as monitor all the reaction events which identify whether a neutron undergoes a virtual collision during the simulation.

3.6.1 The main cycle

After all the desired values have been inputted and necessary cross-sections imported, we start to iterate the main cycle for calculating the k-effective value. Here we break down what the main loop consists of and what importance every section has.

Firstly, let us go through how the k-effective or multiplication factor is obtained. The method we apply is called the k-eigenvalue method, which is one of the most widely used methods in Monte Carlo reactor physics calculations. It is a very reliable method, if the systems are not far from criticality, meaning we are not dealing with highly heterogeneous geometries with large moderator regions or high enrichment level for example. In that case, kinetic parameters would be off by several orders of magnitude and the flux spectrum would become seriously distorted. In such instances, it is highly advisable to consider alternative methods like the α-eigenvalue method, Method of Characteristics (MOC) or discrete ordinates (S_N). Our simulation however deals with a homogeneous PWR-like unit cell with isotropic scattering (at least for the Monte Carlo method). Since we are modelling a stationary self-sustaining chain reaction, the k-eigenvalue method is similar to solving the eigenvalue form of the neutron transport equation.

In total, the number of loops is based on the number of effective and ineffective cycles that were inputted by the user. Firstly, the weights of neutrons are normalised to ensure that the total weight is consistent with the desired number of neutrons born. This normalisation step is critical for maintaining the overall neutron balance.

```python
weight = (weight / np.sum(weight)) * numNeutrons_born
weight0 = weight.copy()
```

After that the code loops over all the neutrons and checks if they are absorbed or not. If they are not absorbed, the neutrons perform a random walk cycle from emission to their eventual absorption, resulting in either a fission or a capture event. While traversing the unit cell, neutrons are allocated a sampled free path length, a crucial step made through the Woodcock delta-tracking method.

One of the more important aspects of this method is determining whether a collision was virtual or not. Emphasising once again the details from earlier chapters, a virtual collision occurs when a neutron undergoes an interaction without absorption, ensuring the preservation of both incident energy and flight direction. If the collision was not virtual, we start a new block where we give our neutron a new sample direction, which is based on the weak assumption that fission and scattering events are isotropic.

```python
def sample_direction():
    eta = np.pi * np.random.rand()
    nu  = 2.0 * np.pi * np.random.rand()
    dirX = np.sin(eta) * np.cos(nu)
    dirY = np.sin(eta) * np.sin(nu)
    return dirX, dirY
```

After that, the neutron is moved around in 2D space, while ensuring that it stays within the cell boundary walls. The neutron moves around as long as a collision occurs.

```python
def move_neutron(x, y, iNeutron, pitch, freePath, dirX, dirY):
    x[iNeutron] += freePath * dirX
    y[iNeutron] += freePath * dirY
    # If outside the cell, find the corresponding point
    # inside the cell
    while x[iNeutron] < 0:
        x[iNeutron] += pitch
    while y[iNeutron] < 0:
        y[iNeutron] += pitch
    while x[iNeutron] > pitch:
        x[iNeutron] -= pitch
    while y[iNeutron] > pitch:
        y[iNeutron] -= pitch

    return x, y
```

Depending on which region the neutron is located in and what sort of energy it has, it is possible to determine if a collision resulted in a fission, capture or a scattering event and update the necessary arrays and variables. This is also the place where we determine if a random collision resulted in a virtual collision and update the variable if necessary. This loop continues as long as a neutron gets absorbed in some collision event.

```
def perform_collision(virtualCollision, absorbed, SigS, SigA,
                      SigP, SigTmax, iGroup, iNeutron,
                      detectS, weight, fuel_chi):
    SigS_sum = np.sum(SigS)
    SigT = SigA + SigS_sum
    SigV = SigTmax[iGroup[iNeutron]] - SigT

    if SigV / SigTmax[iGroup[iNeutron]] >= np.random.rand():
        virtualCollision = True
    else:
        virtualCollision = False

        if SigS_sum / SigT >= np.random.rand():
            detectS[iGroup[iNeutron]] += weight[iNeutron] / SigS_sum
            iGroup[iNeutron] = np.argmax(np.cumsum(SigS) /      \
                                    SigS_sum >= np.random.rand())
        else:
            absorbed = True
            weight[iNeutron] *= SigP / SigA
            iGroup[iNeutron] = np.argmax(np.cumsum(fuel_chi) >= \
                                                np.random.rand())

    return virtualCollision, absorbed, iGroup, weight, detectS
```

After that loop, a game of Russian roulette is played. Specifically, it is a process of assigning a cutoff threshold and terminating neutrons to control the overall neutron population from increasing too quickly and too greatly. This probability is described by the weight, or rather the importance of an individual neutron. If the weight value is greater or equal to some random number, it is killed, otherwise the weight is restored to its previous value.

```
def russian_roulette(weight, weight0):
    numNeutrons = len(weight)
    for iNeutron in range(numNeutrons):
        terminateP = 1 - weight[iNeutron] / weight0[iNeutron]
        if terminateP >= np.random.rand():
            weight[iNeutron] = 0  # killed
        elif terminateP > 0:
            weight[iNeutron] = weight0[iNeutron]  # restore the weight
    return weight
```

Once that is done, a simple check is made to clean off absorbed or killed neutrons. Essentially this function refines the neutron data by removing neutrons with zero weight, meaning they have negligible influence on the simulation. The neutron's positions, energy group and weight arrays are updated if necessary ensuring computational efficiency by focusing on influential neutrons, thereby streamlining the simulation and conserving computational resources.

```
def update_indices(x, y, iGroup, weight):
    # Get the indices of non-zero weight
    indices = np.nonzero(weight)[0]

    # Perform indexing
    x = x[indices]
    y = y[indices]
    iGroup = iGroup[indices]
    weight = weight[indices]

    # Update numNeutrons
    numNeutrons = weight.shape[0]
    return x, y, iGroup, weight, numNeutrons
```

The second to last important process involves splitting “heavy” neutrons in order to release multiple new neutrons into the system. The function responsible for this manages neutron behaviour when the weight exceeds one, which in a sense signifies a gain in potential energy. A random check determines whether a split occurs or not. When it does, new neutrons are released into the system and the function appends them the necessary properties (position, weight and energy group) to dynamically adjust the neutron population.

```
def split_neutrons(weight, numNeutrons, x, y, iGroup):
    numNew = 0
    for iNeutron in range(numNeutrons):
        if weight[iNeutron] > 1:
            N = int(weight[iNeutron])
            if weight[iNeutron] - N > np.random.rand():
                N += 1
            weight[iNeutron] = weight[iNeutron] / N
            for iNew in range(N - 1):
                numNew += 1
                x = np.append(x, x[iNeutron])
                y = np.append(y, y[iNeutron])
                weight = np.append(weight, weight[iNeutron])
                iGroup = np.append(iGroup, iGroup[iNeutron])
    numNeutrons += numNew
    return weight, numNeutrons, x, y, iGroup
```

Finally, after we have split too heavy neutrons it is time to complete the last step in the main cycle, which is to calculate the effective multiplication factor k_{eff} during each cycle of the Monte Carlo simulation. It is computed as the ratio of the sum of the current weights to the sum of weights at the previous step. Additionally, the function distinguishes between inactive and active cycles. For inactive cycles, it prints information about k_{eff} and the number of neutrons. However, during active cycles, additional details such as the mean value of k_{eff} and its standard deviation are calculated and displayed. This is done as long as all the cycles inputted by the user have been completed. In conclusion, this marks the end of the main power cycle and is then followed by importing the data into a HDF5 file and generating PDF plots about the k_{eff} number during each cycle and generating a neutron flux per unit lethargy plot. This officially marks the end of the Monte Carlo code for solving the criticality problem for a steady state Boltzmann neutron transport equation.

3.7 The Discrete Ordinates Method Simulation

The second method of approximating the k-effective value based on the Boltzmann transport equation is the S_N method or the discrete ordinates method. Compared to the Monte Carlo approach, the S_N method offers an alternative, more analytical approach to numerically solve the k-eigenvalue problem. Instead of discretizing the motion of the neutron, the basic idea instead is to discretize a finite number of angles the neutron might take.

The start of the discrete ordinates code begins quite similarly as it did with the Monte Carlo code. This means we import the fuel, cladding and coolant cross-section values from the HDF5 files along with the structural data on reactor geometries. Since we are solving the neutron transport equation in 2D, we need to define a simple 2D grid for our material nodes. Each material is assigned a specific value to show their location in the grid. In our case, "0" is for coolant, "1" is for cladding and "2" is for fuel. An example of this will be presented in the paragraph explaining the final results of the code.

Up next comes the crucial part of defining the N number of points that will henceforth be used in multiple sections of the code, from defining the number of Legendre moments for calculating the scalar flux. Our first order of business is to use it in a function named `getLebedevSphere()`, which is translated from a MATLAB function with the same name and is around 5000 lines long in order to calculate the quadrature weights and roots of the polynomials up to the 131st order. We do not however require such a high degree of accuracy, but it is there for the user to choose and for the potential use for more powerful hardware. [37, 38]

In neutron transport simulations, especially when using the discrete ordinates method, it is common to model reflective boundary conditions. Neutrons that reach a boundary and are not absorbed or leaked are often reflected back into the domain. Therefore, if we wish to identify the corresponding directions in which the neutrons are reflected back from the boundary, we have to cycle through all of the discrete ordinate vectors and compare them. This means writing a statement to check whether the current ordinate $\mu_\mathbf{n}$ is approximately equal to the negation of another ordinate $-\mu_\mathbf{m}$. If a match is found for the X, Y or Z direction, an index for the matching ordinate is stored in a vector for the respective direction.

In the consequent section of the code, the user can change the nature of the scattering source. Specifically, it is a choice to truncate the moment of scattering at the zeroth or first order and define the spherical harmonics for our solver. Spherical harmonics are a way to express functions defined on the surface of a sphere using a series of special mathematical functions. The zeroth moment corresponds to isotropic scattering, where the first is for anisotropic scattering. It goes without saying that there are higher order moments that can be considered to achieve a more accurate result for a real-life case. In the current thesis, that is not the main goal, which we are after. Rather, it is to show a simple example, where we can compare two separate cases. Furthermore, a typical PWR is quite a large scale reactor, where the energy output can vary from a few hundred megawatts to over a gigawatt. For such reactors, modelling leakage for a simplified case, which can occur from scattering events, it is not entirely necessary to model the moments up to such a high degree of accuracy. However, when considering a smaller modular reactor (SMR), a higher degree for scattering moments are necessary, because smaller changes in leakage have a more significant impact on a small reactor. So for our case the zeroth and first order degrees are good enough.

(a) $l = 0, m = 0$ (b) $l = 1, m = 0$

Figure 4. Spherical harmonics for the 0-th (left) and 1-st order (right).

The last two notable sections before the main iteration loop are concerned with constructing the cross-sections materials in our grid and counting the number of equations we have to solve. For cross-section we refer to the 2D grid that we defined above where each material node has a corresponding number ranging from 0 to 2. A nested loop checks the material in each node and populates it with the corresponding macroscopic cross-sections from the imported HDF5 files. The latter segment, as mentioned before, calculates the total number of equations to be solved in the neutron transport simulation. It is done by taking into account the spatial grid, discrete ordinates and certain conditions associated with the boundary and angular quadrature. This number serves a crucial importance for properly sizing the solution vector and matrices in the main iteration loop.

3.7.1 The main cycle

At the start of the main iteration cycle, which calculates the multiplication factor, we define two empty lists for our k-effective values and the residual error, along with the maximum number of cycles the code will run and an initial guess for the system solver. During the cycle, new values will be appended into the two lists and after the solver has converged on a solution, they will be exported and used for plotting. If however our solver does not converge on a specific solution, it is necessary to define a maximum integer number that will stop the iteration process and block the cycle from going on forever.

After copying over the solution vector into our main iteration cycle, the first important step for us is to convert the 1D guess vector to a 4D array to describe the angular flux. This array of arrays depends on the number of energy groups, discrete ordinates and the material cells we defined at the beginning. Additionally, boundary conditions are implemented to ensure physical consistency in the angular flux distribution.

Using our guess for reshaped angular flux, we use it to calculate the according Legendre moment of angular and scalar flux. The formula for this is shown here:

$$\phi^g_{l,m} = \sum_{i=1}^{N} \psi^g \left(\theta_i, \varphi_i\right) w_i R_{l,m} \left(\theta_i, \varphi_i\right) \tag{3.2}$$

This equation expresses the Legendre moment of flux $\phi^g_{l,m}$ as a sum over the product of angular flux ψ^g at discrete angles (θ_i, φ_i), Lebedev grid weights w_i and spherical harmonics $R_{l,m}\left(\theta_i, \varphi_i\right)$. The letter l is the corresponding moment for spherical harmonics we defined above. In code it can be written like so:

```
def calc_fiL_FI(fi, g_nNodesX, g_nNodesY, g_L, g_N, g_R, g_W):
    ng = 421
    fiL = np.zeros((g_nNodesX, g_nNodesY, ng, g_L + 1, 2 * g_L + 1))
    FI =  np.zeros((g_nNodesX, g_nNodesY, ng))

    for iy in range(1, g_nNodesY + 1):
        for ix in range(1, g_nNodesX + 1):
            # convert angular flux fi to the Legendre moments fiL
            for jLgn in range(g_L + 1):
                for m in range(-jLgn, jLgn + 1):
                    SUM = np.zeros(ng)
                    for n in range(g_N):
                        SUM += fi[:, n, ix - 1, iy - 1] * \
                        g_R[n][jLgn, jLgn + m] * g_W[n]
                    fiL[ix - 1, iy - 1, :, jLgn, jLgn + m] = SUM
            # Scalar flux
            FI[ix - 1, iy - 1, :] = fiL[ix - 1, iy - 1, :, 0, 0]

    return fiL, FI
```

Using the scalar flux from this function we can calculate and print out the k-effective value for our reactor. This value can be easily calculated from the ratio of the total production and absorption rate in each node. The corresponding code for it is shown below:

```python
def calculate_keff(SigP, Sig2, FI, volume, SigA, nNodesY, nNodesX):
    pRate = 0
    aRate = 0

    for iy in range(nNodesY):
        for ix in range(nNodesX):
            pRate += (SigP[ix, iy] + 2 * np.sum(Sig2[ix, iy, :], \
                        axis = 1)) @ FI[ix, iy] * volume[ix, iy]
            aRate += SigA[ix, iy] @ FI[ix, iy] * volume[ix, iy]

    keff_value = pRate / aRate

    return keff_value
```

Right after this value is calculated, it is printed out in the terminal along with the corresponding iteration number.

Following this, we have to take these following steps to define the right-hand side of the transport equation equation. Firstly we show how scattering is handled for a specific spatial grid point and then move onto a more general formula that also takes into account fission and other neutron sources. The formula to calculate this can be written as

$$q_S^{g\to g'}(\theta_i, \varphi_i) = \Sigma_S^{g\to g'}\phi^g(\theta_i, \varphi_i) = \sum_{l=0}^{L}\frac{2l+1}{4\pi}\Sigma_{S,l}^{g\to g'}\sum_{m=-l}^{+l}\phi_{l,m}^g R_{l,m}(\theta_i, \varphi_i) \tag{3.3}$$

which represents the scattering source from group g to group g' in units of $\mathrm{cm}^{-3}\mathrm{s}^{-1}\mathrm{sr}^{-1}$. In Python, this formula is implemented like so:

```python
def calc_qT_nEq(ix, iy, nEq, qF, q2, g_N, g_nNodesX, g_nNodesY, g_L,
                                g_muX, g_muY, g_muZ, g_R, SigS, fiL):
    ng = 421
    qT_list_tmp = []

    for n in range(g_N):
        if g_muZ[n] >= 0 and not ((ix == 0 and g_muX[n] > 0) or
                                   (ix == g_nNodesX-1 and g_muX[n] < 0) or
                                   (iy == 0 and g_muY[n] > 0) or
```

```
                          (iy == g_nNodesY-1 and g_muY[n] < 0)):
            # Scattering source (1/s-cm3-steradian),
            # isotropic (g["L"] = 0) or anisotropic (g["L"] > 0)
            qS = np.zeros(ng)
            for jLgn in range(g_L + 1):
                SUM = np.zeros(ng)
                for m in range(-jLgn, jLgn + 1):
                    SUM += fiL[ix, iy][:, jLgn, jLgn + m] * \
                           g_R[n][jLgn, jLgn + m]
                qS += (2*jLgn+1) * np.dot(np.transpose(
                        SigS[jLgn][ix, iy]), SUM) / (4*np.pi)

            nEq += 1
            # Right-hand side is a total neutron source:
            qT_list_tmp.append(qF + q2 + qS)

    return qT_list_tmp, nEq
```

To get the total sources on the right-hand side, we have to combine three different source formulas, which take into account scattering, fission and neutrons gained from a (n,2n) reaction. All those sources are added up with this function:

```
def compute_qT(FI, fiL, chi, keff, SigP, Sig2, SigS, g_N, g_L, g_R,
                         g_muX, g_muY, g_muZ, g_nNodesX, g_nNodesY):
    nEq = 0
    ng = int(421)
    q2 = np.zeros(ng)
    qT_list = []

    for iy in range(g_nNodesY):
        for ix in range(g_nNodesX):
            # Fission source (1/s-cm3-steradian)
            qF = matmul(chi[ix, iy], SigP[ix, iy]) @ FI[ix, iy] /   \
                    keff[-1] / (4*np.pi)

            # Isotropic source from (n,2n) (1/s-cm3-steradian)
            q2 = 2 * np.dot(np.transpose(Sig2[ix, iy]), FI[ix, iy]) \
                    / (4*np.pi)
            qT_list_tmp, nEq = calc_qT_nEq(ix, iy, nEq, qF, q2, g_N,
                    g_nNodesX, g_nNodesY,g_L, g_muX, g_muY, g_muZ,
                    g_R, SigS, fiL)

            # Right-hand side is a total neutron source:
            for ii in range(len(qT_list_tmp)):
                qT_list.append(qT_list_tmp[ii])

    return qT_list
```

Using this function leaves our values for the right-hand side not in a vector, but in a 2D array. Our goal however is to have it in a 1D vector format, which would represent b in formula $Ax = b$. Therefore our final step for the right-hand side of the equation is to reshape our 2D matrix into a 1D vector. This marks the end for how the solution values on the right-hand side are constructed.

Up next we will be setting up the left-hand side of the equation, which is composed of the advection term (leakage) and the absorption term. For the advection formulas we need to discretize first order derivatives for each direction of the discrete ordinate vector. This leaves us with the finite difference equation

$$\mu \cdot \nabla\psi = \mu_x \frac{\psi_{i,j}^{g,N} - \psi_{i-1,j}^{g,N}}{\Delta x} + \mu_y \frac{\psi_{i,j}^{g,N} - \psi_{i,j-1}^{g,N}}{\Delta y} \tag{3.4}$$

In the actual code however these gradients are applied in either the forward or backwards difference scheme to account for the boundaries of the matrices and the overall structure of the angular flux:

```
def grad(n, ix, iy, fi, muX, muY, nRefX, nRefY, nNodesX, nNodesY,
                                                            delta):
    if muX[n] > 0:
        if ix == 0:
            dfiX = fi[:, nRefX[n] - 1, ix, iy] - \
                   fi[:, nRefX[n] - 1, ix + 1, iy]
        else:  # if ix > 0
            dfiX = fi[:, n, ix, iy] - fi[:, n, ix - 1, iy]
    else:  # if muX(n) <= 0
        if ix == nNodesX - 1:
            dfiX = fi[:, nRefX[n] - 1, ix - 1, iy] - \
                fi[:, nRefX[n] - 1, ix, iy]
        else:  # if ix < nNodesX - 1
            dfiX = fi[:, n, ix + 1, iy] - fi[:, n, ix, iy]

    if muY[n] > 0:
        if iy == 0:
            dfiY = fi[:, nRefY[n] - 1, ix, iy] - \
                fi[:, nRefY[n] - 1, ix, iy + 1]
        else:  # if iy > 0
            dfiY = fi[:, n, ix, iy] - fi[:, n, ix, iy - 1]
    else:  # if gmuY(n) <= 0
        if iy == nNodesY - 1:
            dfiY = fi[:, nRefY[n] - 1, ix, iy - 1] - \
                fi[:, nRefY[n] - 1, ix, iy]
        else:  # if iy < nNodesY - 1
            dfiY = fi[:, n, ix, iy + 1] - fi[:, n, ix, iy]

    dfidx = dfiX / delta
    dfidy = dfiY / delta

    return dfidx, dfidy
```

Finally we define the left-hand side of the transport equation, specifically the matrix A, which has a main diagonal that is populated with the values we get by summing up the gradient terms with the absorption term, which is just the total macroscopic cross section times the angular flux.

```
def calculate_LHS(solution, N, nNodesX, nNodesY, muX, muY, muZ,
                              nRefX, nRefY, nRefZ, delta, SigT):
    # Convert 1D solution vector x to the cell array
    # of angular flux fi
    fi = reshapeAngFlux(solution,
                        N,      nNodesX,  nNodesY,
                        muX,    muY,      muZ,
                        nRefX,  nRefY,    nRefZ)

    nEq = 0
    LHS_list = []
```

```
for iy in range(nNodesY):
    for ix in range(nNodesX):
        for n in range(N):
            if muZ[n] >= 0 and not ((ix == 0 and muX[n] > 0) \
                or (ix == nNodesX - 1 and muX[n] < 0) or      \
                  (iy == 0 and muY[n] > 0) or                 \
                  (iy == nNodesY - 1 and muY[n] < 0)):
                # Gradients
                dfidx, dfidy = grad(n, ix, iy, fi, muX, muY,
                nRefX, nRefY, nNodesX, nNodesY, delta)
                nEq += 1
                LHS_list.append(muX[n] * dfidx + muY[n] *     \
                dfidy + SigT[ix, iy] * fi[:, n, ix, iy])

# Make 1D vector
LHS = np.zeros(len(LHS_list) * len(LHS_list[0]))

for i in range(len(LHS_list)):
    for j in range(len(LHS_list[i])):
        LHS[i * len(LHS_list[i]) + j] = LHS_list[i][j]

return LHS
```

Now we can write out the final transport equation for the discrete ordinates method that is used to create a linear system of equations for the form $Ax = b$.

$$\left(\mu_x \frac{\psi_{i,j}^{g,N} - \psi_{i-1,j}^{g,N}}{\Delta x} + \mu_y \frac{\psi_{i,j}^{g,N} - \psi_{i,j-1}^{g,N}}{\Delta y}\right) + \Sigma_t \cdot \psi^{g,N} =$$
$$= \sum_{l=0}^{L} \frac{2l+1}{4\pi} \Sigma_{S,l}^{g \to g'} \sum_{m=-l}^{+l} \sum_{i=1}^{N} \phi^g(\mu_i) W_i R_{l,m}(\mu_i) R_{l,m}(\mu_i) +$$
$$+ \frac{1}{k_{\text{eff}}} \cdot \frac{1}{4\pi} \cdot \chi \cdot \Sigma_P \cdot \phi + \frac{2}{4\pi} \cdot \Sigma_{(n,2n)} \cdot \phi \tag{3.5}$$

The left hand side of the equation will be rewritten into the matrix A and the right-hand side will equate to the vector b.

To solve this system of equations in the form $Ax = b$, we use the aforementioned BiCGSTAB solver, but not the standard one found in the SciPy package library. It is a good implementation, but as it is with most things found in the SciPy package, they are incredibly slow. This is why we will be using a specialised BiCGSTAB solver originally meant for MATLAB, written by C. T. (Tim) Kelley. [39] His solver is easy to implement and translate into Python. Furthermore, it allows us to write a more tailored version of this function or optimise it for faster implementation.

This function returns us three things. Firstly, the computed solution vector, which will be used at the start of the next iteration cycle. Secondly, a list of error values, which are used to calculate the residual error $||b - A \cdot x||$ to reveal how close the algorithm is to converging for a given value of x. And thirdly, the total number of iterations performed for each time the function is called. All these values are then printed next to the k-effective value for the current iteration cycle. The final step of the main iteration cycle is to check whether the number of inner cycles, meaning the cycles the BiCGSTAB solver goes through, is equal to zero. If true, the computations are finished, meaning we have reached our desired level of accuracy for the N number of discrete ordinates that we chose for our simulations.

The final steps of the discrete ordinates code is to output the results into a HDF5 file and plot the results. The added benefit of using the discrete ordinate method is that we also gather information about the flux. Therefore we can visualise the flux in the thermal, resonance and fast zones in each unit cell of the material matrix that we constructed. In addition we can also visualise the neutron flux in each unit cell and steps the BiCGSTAB solver took to reach its final value. This officially marks the end of the discrete ordinates method or S_N method.

4. Results

In this section we will make a short demonstration how the Monte Carlo and Discrete Ordinates codes work. This meant for the user to familiarise themselves with what they can change and toggle in the code in order to model the pressurised water reactor under specific conditions with varying levels of numerical accuracy.

Before going into detail on what specific inputs each code has, let us first bring out the common element in both of the code, which is importing the macroscopic cross-sections. For this we use a special function which accepts info on the material type and the temperature value in Kelvin. Using these inputs, the function searches for the appropriate HDF5 file from the `02.Macro.XS.421g/` folder and converts it into a local Python dictionary, making it easy to access different types of data. Therefore, before starting the MC or SN code, we must first create the appropriate macroscopic cross-section files for the respective temperature conditions we wish to simulate. At the beginning of the `createUO2_03.py`, `createH2OB.py` and `createZry.py` there is a variable that the user can change which allows us to construct a macroscopic cross-section file from the appropriate microscopic cross-sections. For our examples below, we will choose the fuel temperature to be 900 K, while cladding and coolant are both 600 K.

4.1 Monte Carlo code: inputs and results

Starting off with the Monte Carlo code, we first define some general values about our reactor. These include values such as the pitch, the height of the square unit cell, and the boundary sizes which define the fuel and coolant regions. Following this we set the number of starting neutrons in the system that are randomly distributed throughout the unit cell. This variable is defined as `numNeutrons_born` and for this example is set to 100 neutrons. After that we define the number of inactive cycles. Recalling the explanations in previous chapters, it is the number of cycles the solver skips before it starts collecting data. This gives the system time to reach a steady state. For the current case, we set this variable, named `numCycles_inactive`, equal to 100.

```
# Number of source neutrons
numNeutrons_born = 100        # INPUT

# Number of inactive source cycles to skip before
# starting k-eff accumulation
```

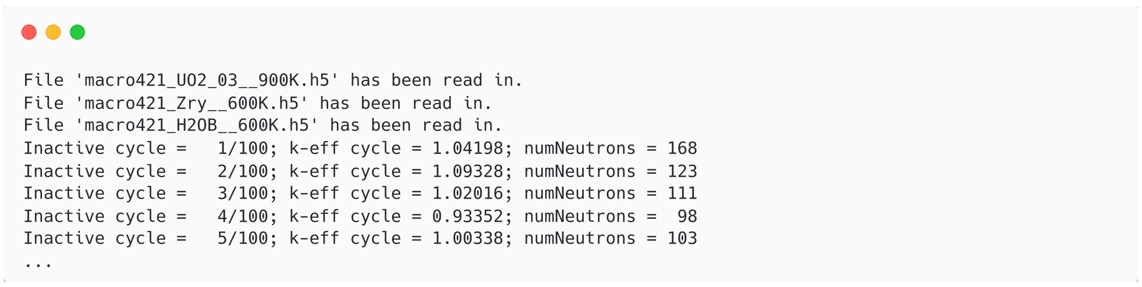
```
numCycles_inactive = 100     # INPUT

# Number of active source cycles for k-eff accumulation
numCycles_active = 2000      # INPUT

# Size of the square unit cell
pitch = 3.6  # cm            # INPUT

# Define fuel and coolant regions
fuelLeft, fuelRight = 0.9, 2.7  # INPUT
coolLeft, coolRight = 0.7, 2.9  # INPUT
```

Starting off with our code, the output in the terminal will look like this:

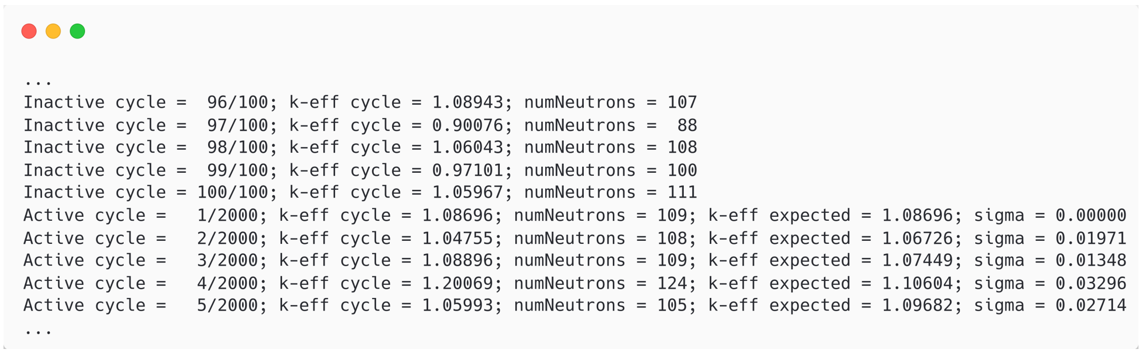


```
File 'macro421_UO2_03__900K.h5' has been read in.
File 'macro421_Zry__600K.h5' has been read in.
File 'macro421_H2OB__600K.h5' has been read in.
Inactive cycle =   1/100; k-eff cycle = 1.04198; numNeutrons = 168
Inactive cycle =   2/100; k-eff cycle = 1.09328; numNeutrons = 123
Inactive cycle =   3/100; k-eff cycle = 1.02016; numNeutrons = 111
Inactive cycle =   4/100; k-eff cycle = 0.93352; numNeutrons =  98
Inactive cycle =   5/100; k-eff cycle = 1.00338; numNeutrons = 103
...
```

Figure 5. Snapshot from the terminal, which shows the start of the Monte Carlo solver.

As we can see from the terminal output above, we have read in three files for our fuel, cladding and coolant and started the loop of inactive cycles. The second column, which prints out “k-eff cycle” is the multiplication factor in the current neutron generation cycle and `numNeutrons` is the current number of neutrons within the system. After the inactive cycles have finished, we start our active cycles for data accumulation. This means that we also need to input the number of active cycles named `numCycles_active` right after inactive ones.

However, choosing the sufficient number of cycles is not exactly quite well defined. According to the law of large numbers, as the number of simulation cycles increases, the situated result should converge towards the true or expected value. Although, the rate of convergence may vary depending on the characteristics of the system and simulation algorithm. Through trial and error, it was determined that a sufficiently large size for most simulations is around 2000 active cycles. Nevertheless, one should keep in mind that the process is stochastic. Most times the simulation converged on the final solution quite fast with a minimal relative error, but there were a few times, where even after 2000 active cycles, the standard deviation from the expected k-effective value was rather large.



```
...
Inactive cycle =  96/100; k-eff cycle = 1.08943; numNeutrons = 107
Inactive cycle =  97/100; k-eff cycle = 0.90076; numNeutrons =  88
Inactive cycle =  98/100; k-eff cycle = 1.06043; numNeutrons = 108
Inactive cycle =  99/100; k-eff cycle = 0.97101; numNeutrons = 100
Inactive cycle = 100/100; k-eff cycle = 1.05967; numNeutrons = 111
Active cycle =   1/2000; k-eff cycle = 1.08696; numNeutrons = 109; k-eff expected = 1.08696; sigma = 0.00000
Active cycle =   2/2000; k-eff cycle = 1.04755; numNeutrons = 108; k-eff expected = 1.06726; sigma = 0.01971
Active cycle =   3/2000; k-eff cycle = 1.08896; numNeutrons = 109; k-eff expected = 1.07449; sigma = 0.01348
Active cycle =   4/2000; k-eff cycle = 1.20069; numNeutrons = 124; k-eff expected = 1.10604; sigma = 0.03296
Active cycle =   5/2000; k-eff cycle = 1.05993; numNeutrons = 105; k-eff expected = 1.09682; sigma = 0.02714
...
```

Figure 6. Snapshot from the middle of Monte Carlo iteration cycles. Here we can see, how the output in the terminal switches when the number of inactive cycles has finished and the active cycles have started. The two additional columns, named “k-eff expected” and “sigma”, represent the mean k-effective value and the standard deviation from the mean respectively.

After all the active cycles have finished, we can use the k-effective expected values together with the standard deviation to plot the graph of the average multiplication factor for each neutron generation. As we can see from the plot, the current simulation converged on an expected solution quite quickly, leaving the standard deviation to be quite small. This is more or less what the ideal graphic should look like. Furthermore, given the current inputs that we gave on our simulation, the average expected value for k-effective came out around 1.05, meaning the reactor is 5% supercritical. To lower this value, we could design our reactor unit cell to have larger cladding zones or fuel cells with a smaller pitch size. For example, changing the pitch value to be a centimetre smaller results in the k-effective value to be around 10% lower, making the reactor sub-critical. However, if we were to increase the cladding zones to be 0.4 cm thick instead of 0.2 cm on both sides, it would give us roughly a 6% decrease, which is around the ideal value our reactor should be working at. Plots of these additional changes can be found under Appendix 2.

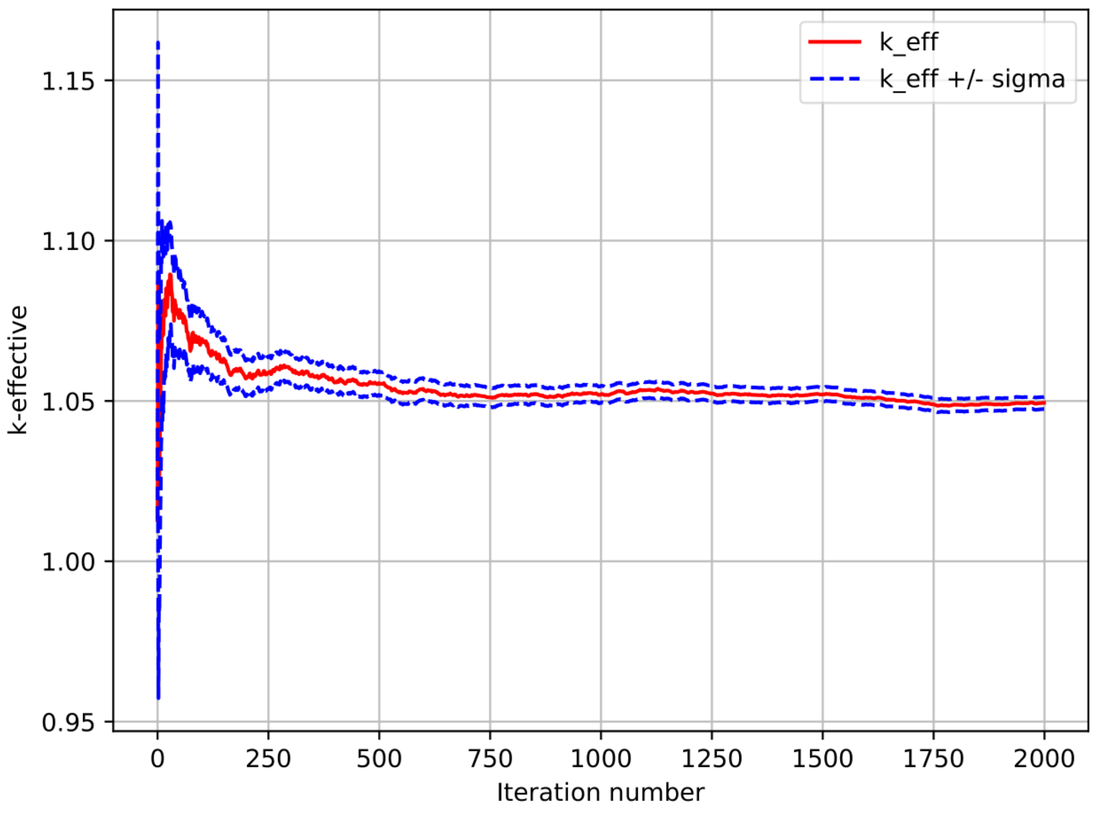


Figure 7. Graphical output of the k-effective value over the number of iteration cycles for the Monte Carlo method.

Since we do not gather information on surface crossings with the Monte Carlo method, we use the concept of lethargy to estimate the neutron flux based on detected scattering events. Neutron lethargy, denoted by the dimensionless logarithm u, represents the logarithmic energy decrement. It is defined as the natural logarithm of the ratio of the energy source neutrons (E_0) to the energy of neutrons after a collision (E): $u = \ln(E_0/E)$. When visualising E against u in a curve, where the

$$E = E_0 \cdot \exp(-u) \tag{4.1}$$

plot reveals an exponential decay of energy per unit collision. This decay illustrates that the most significant energy reductions occur during the initial collisions. In essence, this representation emphasises the concept that if we wish for our fission decay neutrons to transition to thermal neutrons, it is crucial to confine them. This confinement allows the neutrons to undergo multiple collisions, dissipating energy before exiting the region of interest. Using this information, we can use our energy group data from fuel's macroscopic cross-sections and compare it to our neutron energies after scattering events. Using this ratio, we can visualise the lethargy plot, which is shown here:

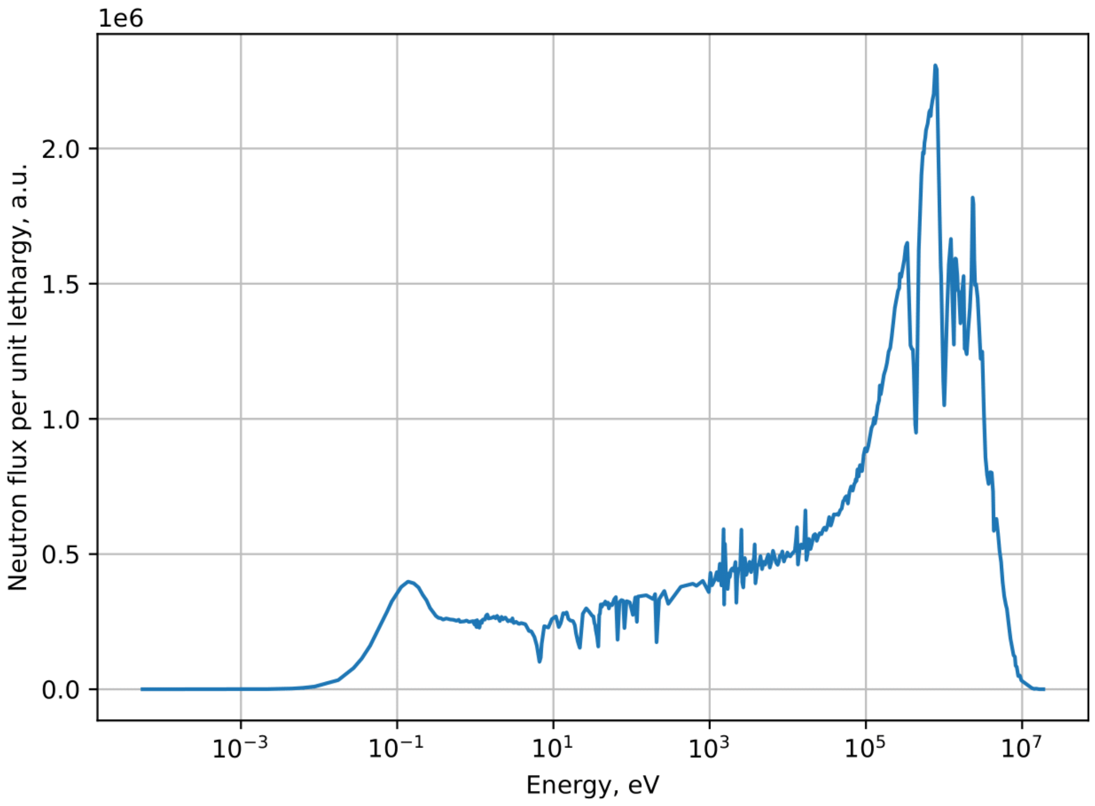


Figure 8. Graph of the neutron flux per unit lethargy for fuel from the Monte Carlo simulation.

4.2 Discrete Ordinates code: inputs and results

Switching over to the discrete ordinates code, our first order of business is to define the number of nodes for the 2D material matrix. This is the matrix that is going to define the layout for our fuel, cladding and coolant placement. For our example, we will design our 2D material matrix to look like this:

```
# Define the material for each node
# (0 is coolant, 1 is cladding, 2 is fuel)        # INPUT
mat = np.array([[2, 2, 2, 2, 2, 1, 0, 0, 0, 0],
                [2, 2, 2, 2, 2, 1, 0, 0, 0, 0]])
```

We define a material for each node as described in the previous chapter: “0” is for coolant, “1” is for cladding and “2” is for fuel. The main purpose of this is to help us construct the cross-sections for each material and after the main iteration cycle define the flux in each node. Since the placement of the materials in the matrix is important as well, it is also necessary to specify the number of nodes in X and Y direction. Currently, our material matrix has ten rows and two columns.

Following this, we can define the most important number in the code, which is the number of discrete ordinates of the Lebedev-Laikov quadrature grid-points. This is the main component that we use for calculating surface integrals in our code. However, it is important to remember that the higher the degree (meaning order), the higher the accuracy, but the longer we also have to wait for the computations to finish. In our current demo, we will choose the 17th order for our polynomials, which matches to the 110th degree for input.

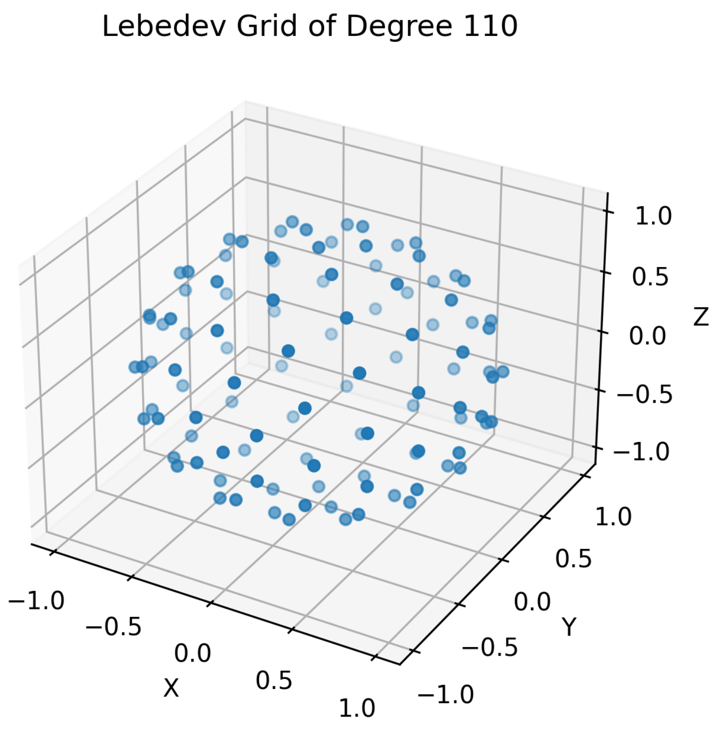


Figure 9. 3D plot visualising 110 of the discrete ordinates on a unit sphere generated by the Lebedev-Laikov quadrature function.

Going forward from here we can define the scattering source anisotropy. As explained in the previous chapter, we only look at two simplified cases. Firstly, when our scattering source is isotropic, we give an integer valued input “0”, if it is anisotropic, we input “1”. For our demo, let us choose anisotropic scattering. After we have chosen our input, we must calculate the spherical harmonics for each ordinate. In the code, it can be summed up like this:

```
# Scattering source anisotropy: 0 — P0 (isotropic),
# 1 — P1 (anisotropic)
g['L'] = 1  # INPUT
# Initialize the 'R' key in the 'g' dictionary
g['R'] = np.zeros((g['N'], int(2*g['L'] + 1), int(2*g['L'] + 1)))
# Calculate spherical harmonics for every ordinate
for n in range(g['N']):
    for jLgn in range(g['L'] + 1):
        for m in range(-jLgn, jLgn + 1):
            if jLgn == 0 and m == 0:
                g['R'][n][jLgn, jLgn + m] = 1
            elif jLgn == 1 and m == -1:
                g['R'][n][jLgn, jLgn + m] = g['muZ'][n]
            elif jLgn == 1 and m == 0:
                g['R'][n][jLgn, jLgn + m] = g['muX'][n]
            elif jLgn == 1 and m == 1:
                g['R'][n][jLgn, jLgn + m] = g['muY'][n]
```

Final variables we can change are concerned with setting maximum and cutoff parameter values for our iterations. First value named numIter is an integer value, which defines the maximum number of iterations the main outer cycle can undergo before stopping further iterations. This number mainly depends on the patients of the user, but around 200 should be good enough for us. The last two values which are found before the execution of the custom BiGCSTAB solver are `errtol` and `maxit`. These describe relative residual reduction factor (or residual error for short) and the maximum number of allowed iterations respectively. In our current case, `errtol` is set to 10^{-4} and `maxit` is 2000.

This is the output from the terminal at the start of the code. Based on the number of discrete ordinates that we chose, there is a greater number of equations to be solved if the order of polynomials is high. Once the main iteration cycle starts, the first value it outputs is the k-effective value in the current cycle `nOuter`. The `nInner` value is the number of iterations the BiCGSTAB solver needed to reach the required error target value and is outputted as `residual`, together with the target that we chose, which is the input `errtol`. This cycle will last as long as nOuter is equal to 200, or stop iterations preemptively when `nInner` is equal to zero, meaning a steady state has been reached and there is no further difference in k-effective values.

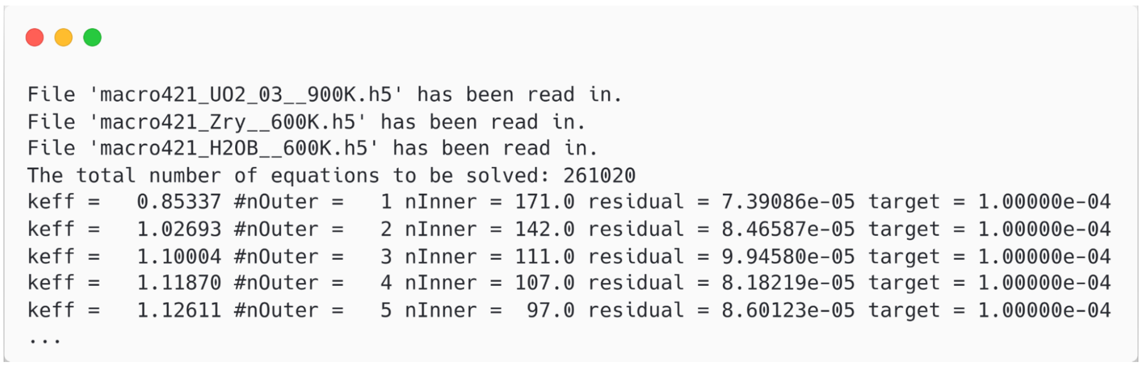

```
File 'macro421_UO2_03__900K.h5' has been read in.
File 'macro421_Zry__600K.h5' has been read in.
File 'macro421_H2OB__600K.h5' has been read in.
The total number of equations to be solved: 261020
keff =   0.85337 #nOuter =   1 nInner = 171.0 residual = 7.39086e-05 target = 1.00000e-04
keff =   1.02693 #nOuter =   2 nInner = 142.0 residual = 8.46587e-05 target = 1.00000e-04
keff =   1.10004 #nOuter =   3 nInner = 111.0 residual = 9.94580e-05 target = 1.00000e-04
keff =   1.11870 #nOuter =   4 nInner = 107.0 residual = 8.18219e-05 target = 1.00000e-04
keff =   1.12611 #nOuter =   5 nInner =  97.0 residual = 8.60123e-05 target = 1.00000e-04
...
```

Figure 10. A snapshot of the terminal from the start of the discrete ordinates code.

After all the iterations have finished, we can plot the steps the BiCGSTAB solver took in order to reach a steady state solution. As you can see from our initial guess for the solver, it initially takes huge steps to converge on a stable solution, but then overshoots and tries to balance it out by underscoring the k-effective value, after which it settles around 1.05. Furthermore, we can also make out that we achieved this solution with roughly 85 steps. Similar to our previous example in Monte Carlo, our current criticality is a little higher than normal. If we do not add any additional rows and columns to our material matrix, we can make a few changes to reduce criticality. For example, we could remove two of the fuel columns in the matrix, replace it with one cladding column and fill all the remaining nodes with coolant values. In short, we would reduce the fuel zone and increase the coolant zone. This would reduce our original criticality value by about 4%. A plot of this can be found under Appendix 3.

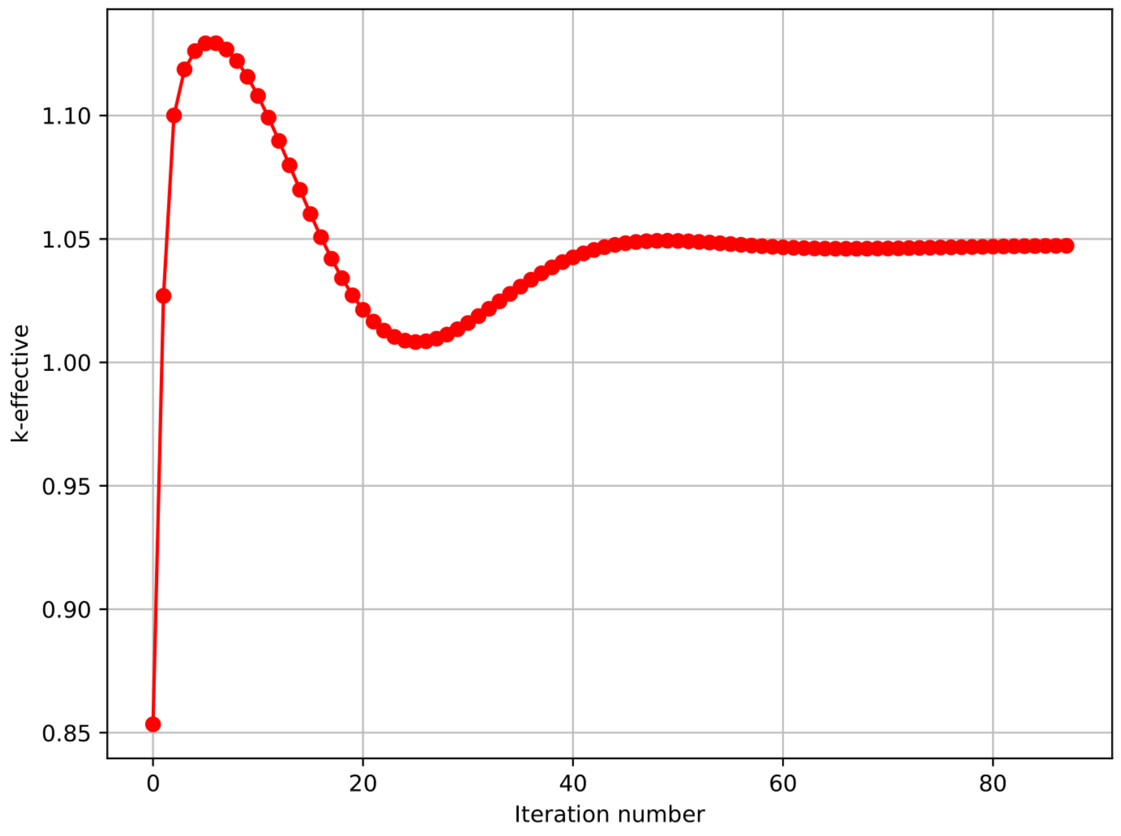


Figure 11. Graphical output of the k-effective value over the number of iteration cycles for the discrete ordinates method.

Calculating the flux lethargy plot is a bit different this time. At the start we define the volume of our material matrix. We can input our own step spacing size to define the volume at each node. In our example, our grid step size is set to 0.2 cm and the nodes at the boundaries only cover half of the volume compared to the interior nodes. Since we calculate the total scalar flux in our code, we can determine the flux for fuel, cladding and coolant by multiplying the total total flux with total volume and dividing it by the volume of the cell in respect to each material node. This steps allows us to visualise the the flux lethargy for each zone:

Here is a plot which describes the lethargy for each material:

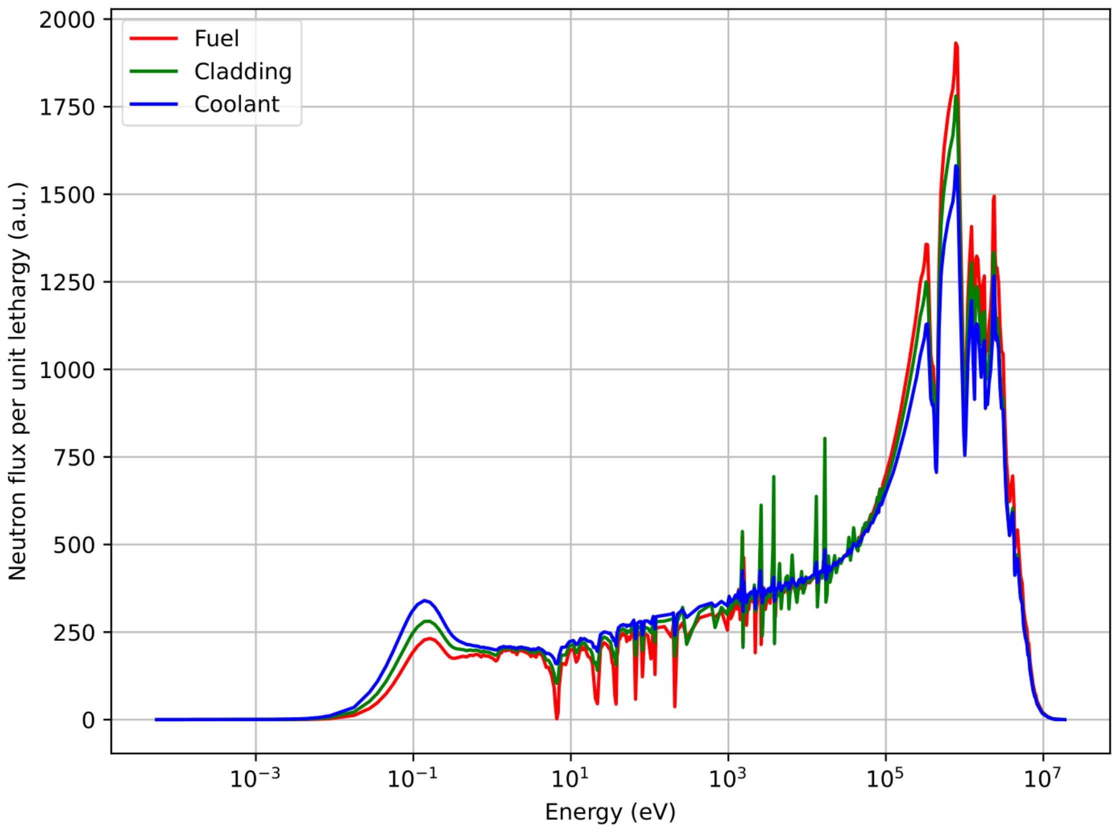


Figure 12. Graph of the neutron flux per unit lethargy for fuel from the discrete ordinates simulation.

For a 421-energy group GENDF file, it is possible to display the thermal, resonance and fast zones separately at certain energy cutoff values. These values are the following: energies below 1 eV fall within the thermal zone, those surpassing 0.1 MeV are categorised in the fast zone and anything between these two thresholds is attributed to the resonance zone. This allows us to visualise the neutron flux in each region.

As we can see from the Figure 13, the flux in the fast region is higher than the rest, because fast neutrons are born from fission reactions inside the fuel cells and have a high kinetic energy. This stays true for each of the fuel cells that we defined with our material matrix. As we move further away from the fuel cells into the coolant nodes, we can see how the flux in the thermal zone starts to increase while neutron flux in the fast zone decreases. This makes sense because fast neutrons lose energy in the coolant zone. They are either absorbed or thermalized, causing an increase in the thermal zone. A similar effect can also be seen for the resonance zone, since the resonance zone becomes more prevalent as the energy of fission neutrons decrease in the coolant nodes.

There are several additional plots that we can visualise with the data that we have. These include the residual error for every iteration step that we have or the fast, resonance and thermal flux distribution for every cell of our material matrix. All of these additional graphs can be found under Appendix 3.

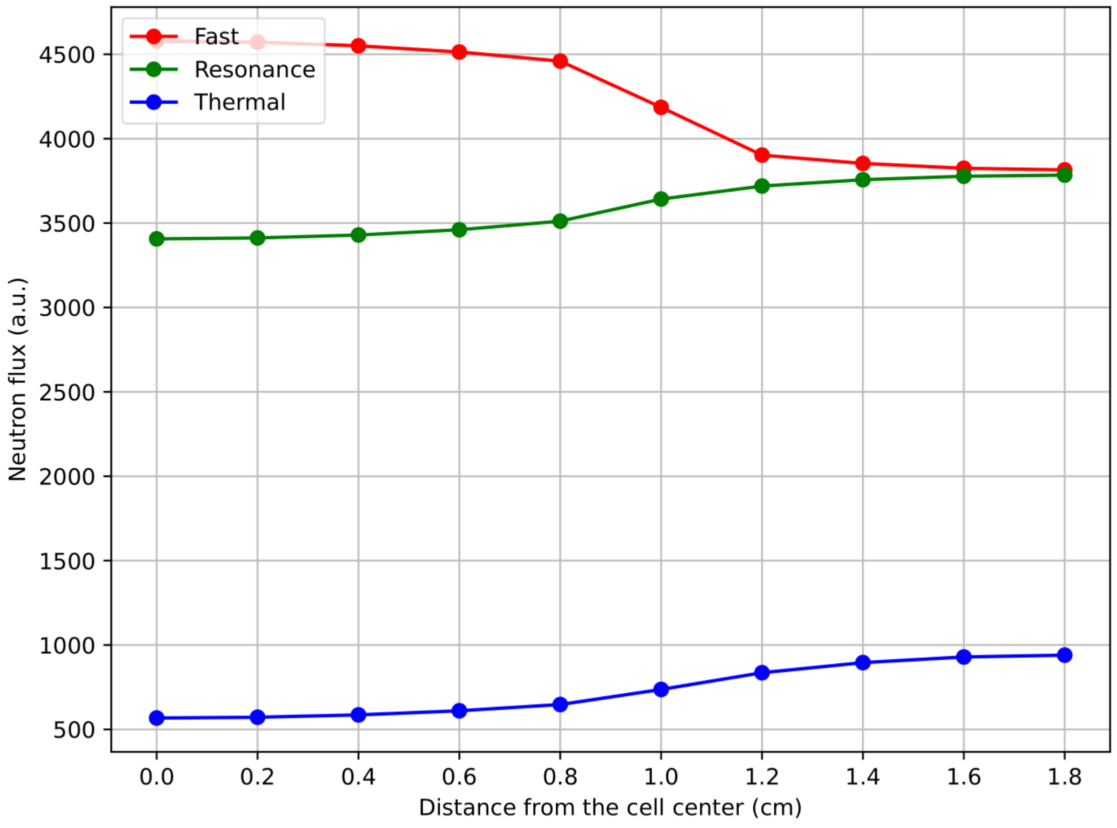


Figure 13. Graph of the scalar neutron flux per node for fast, resonance and thermal regions. Each point in the graph specifies the physical center of an element from the material matrix.

5. Summary

The thesis at hand has successfully investigated and implemented simplified neutron transport modelling using two distinct methods, Monte Carlo and discrete ordinates. Both methods are applicable for addressing the Boltzmann transport equation in 2D, but they employ distinct strategies for solving the steady-state formula.

Every step of the process was documented and presented with the necessary theoretical background. Starting with the gathering of initial data from the IAEA website to solving it with two of the most popular methods. The approach to solving the Boltzmann transport equation can either be stochastic or deterministic in nature. Both methods are implemented from scratch, presenting a simplified view to solve the transport equation using distinct numerical methods.

The Monte Carlo is a stochastic approach, which offers an efficient way to simplify geometry routines and accelerate calculations in intricate geometries. This is done by applying the Woodcock delta-tracking method to model the behaviour of neutrons in a unit cell. While it does speed up the calculations to converge on the true solution, it is not often applied by most high-level Monte Carlo codes, which rely on complicated ray-tracing algorithms to model the movement of neutrons. Furthermore, future derivations of this work can use this code as a reference point or offer further optimization techniques, such as doing parallel computations for an even faster implementation.

The discrete ordinates method demonstrates superior computational accuracy for solving surface integrals that are mainly required for modelling scattering events. These surface integrals are calculated by the use of Lebedev quadratures, which are known for their high level of accuracy for calculating surface integrals. Additionally, it can be shown that the scattering source can be either isotropic or anisotropic. However, high orders of spherical harmonics are necessary to accurately model neutron flux in smaller reactors. Furthermore, a higher number of discrete ordinates pose a significant cost to the computational resources, which can most likely be solved by doing parallel computations on a GPU.

The Boltzmann transport equation is notorious for being one of the most difficult equations to solve in physics. This thesis offers a way to show two practical ways of numerically solving this equation and finding the criticality value of a typical PWR. Further advancements in the code can be pursued by incorporating faster parallel computation techniques.

The code, in its current state, also serves as a valuable resource for future endeavors, such as implementing the α-eigenvalue method, incorporating a simplified constructive solid geometry module or introducing the stochastic version of the discrete ordinates method, which would explore the relationship between both methods.

6. Acknowledgements

I would like to express my deepest and sincerest gratitude to my advisor, Rodrigo Oliveira. He served as my first lecturer, who introduced me into the field of reactor physics. His profound knowledge and extensive experience in the field never ceases to amaze me and serves as a tremendous source of inspiration. Rodrigo provided invaluable support during challenging times, remaining calm and helpful when all my knowledge and hope seemed lost. Moreover, he demonstrated patience and understanding, giving me the time and space to learn and develop the necessary skills on my own. On a personal level, I would never have envisioned myself writing a thesis on such a captivating topic. Throughout this journey, I have learned a lot, but have also gained a profound understanding of the vastness of what there is still to learn. I extend my heartfelt thanks for all the support and the opportunity to delve into such an engaging topic under Rodrigo's guidance.

References

[1] Marshall Holloway and Richard Baker. “Note on the Origin of the Term ’Barn’”. In: *Critical Assembly*. Cited in *Critical Assembly* by Lillian Hoddeson et al., p. 430, footnote 51. Lillian Hoddeson et al., 1944, p. 430.

[2] *Neutron Cross-section*. [Accessed: December 15, 2023]. URL: https://www.nuclear-power.com/neutron-cross-section/.

[3] Jaakko Leppänen. “Development of a New Monte Carlo Reactor Physics Code”. PhD thesis. 2007.

[4] *Neutron Energy*. [Accessed: 15-12-2023]. URL: https://www.nuclear-power.com/nuclear-power/reactor-physics/atomic-nuclear-physics/fundamental-particles/neutron/neutron-energy/.

[5] H. Baer and R.N. Cahn. “Cross-Section Formulae for Specific Processes”. In: (2009).

[6] *Macroscopic Cross-section*. [Accessed: 15-12-2023]. URL: https://www.nuclear-power.com/nuclear-power/reactor-physics/nuclear-engineering-fundamentals/neutron-nuclear-reactions/macroscopic-cross-section/.

[7] Patrick Morilhat et al. “Nuclear Power Plant Flexibility at EDF”. In: *EDF Research & Development* (2019).

[8] International Atomic Energy Agency. *Natural Circulation in Water Cooled Nuclear Power Plants: Phenomena, Models, and Methodology for System Reliability Assessments*. IAEA-TECDOC 1474. Nuclear Power Technology Development Section, 2005.

[9] William M. Haynes, David R. Lide, and Thomas J. Bruno. *CRC Handbook of Chemistry and Physics: A Ready-Reference Book of Chemical and Physical Data*. 97th. Boca Raton, Florida: CRC Press, 2016, p. 734.

[10] *Alloy M5 Cladding Performance Update*. Accessed: 15-12-2023. 2011. URL: https://www.neimagazine.com/features/featurealloy-m5-cladding-performance-update/.

[11] J. J. O’Connor and E. F. Robertson. *Ludwig Boltzmann*. [Accessed: 15-12-2023]. URL: https://mathshistory.st-andrews.ac.uk/Biographies/Boltzmann/.

[12] Philip T. Gressman and Robert M. Strain. “Global Classical Solutions of the Boltzmann Equation with Long-Range Interactions”. In: *PNAS* (2010).

[13] R. J. DiPerna and P. L. Lions. “On the Cauchy Problem for Boltzmann Equations: Global Existence and Weak Stability”. In: *The Annals of Mathematics, Second Series* 130.2 (1989), pp. 321–366.

[14] Tara M. Pandya et al. “Two-Step Neutronics Calculations with Shift and Griffin for Advanced Reactor Systems”. In: *Annals of Nuclear Energy* (2021).

[15] Derek R. Gaston et al. “Method of Characteristics for 3D, Full-Core Neutron Transport on Unstructured Mesh”. In: *Nuclear Technology* 207.7 (2021), pp. 931–953. DOI: 10.1080/00295450.2021.1871995.

[16] C. E. Shannon. “A Mathematical Theory of Communication”. In: *The Bell System Technical Journal* 27 (1948), pp. 379–423, 623–656.

[17] E. R. Woodcock et al. “Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry”. In: ANL-7050 (1965).

[18] Jaakko Leppänen. “Performance of Woodcock Delta-Tracking in Lattice Physics Applications Using the Serpent Monte Carlo Reactor Physics Burnup Calculation Code”. In: *Annals of Nuclear Energy* 37 (2010), pp. 715–722.

[19] Simon D. Richards et al. “MONK and MCBEND: Current Status and Recent Developments”. In: *Annals of Nuclear Energy* 82 (2015), pp. 63–73.

[20] J. Seco and F. Verhaegen. *Monte Carlo Techniques in Radiation Therapy*. 2013.

[21] László Szirmay-Kalos, Balázs Tóth, and Milán Magdic. “Free Path Sampling in High Resolution Inhomogeneous Participating Media”. In: *Computer Graphics Forum* 30.1 (2011), pp. 85–97.

[22] Victor S. Antyufeev. “Mathematical Verification of the Monte Carlo Maximum Cross-Section Technique”. In: *Monte Carlo Methods and Applications* (2015). DOI: 10.1515/mcma-2015-0106.

[23] Shoji Asano. “On the Discrete Ordinates Method for the Radiative Transfer”. In: *Journal of the Meteorological Society of Japan. Ser. II* 53.1 (1975), pp. 92–95. DOI: 10.2151/jmsj1965.53.1_92.

[24] S. Chandrasekhar. *Radiative Transfer*. New York: Dover, 1960, p. 393.

[25] F. R. Mynatt, F. J. Muckenthaler, and P. N. Stevens. “Development of Two-Dimensional Discrete Ordinates Transport Theory for Radiation Shielding”. In: (1969).

[26] Milan Hanuš. “Mathematical Modeling of Neutron Transport”. PhD thesis. 2014.

[27] Bengt Fornberg and Jordan M. Marte. “On Spherical Harmonics Based Numerical Quadrature over the Surface of a Sphere”. In: (2000).

[28] H. A. van der Vorst. “Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems”. In: *SIAM Journal on Scientific Computing* (1992).

[29] International Atomic Energy Agency. *Management of Reprocessed Uranium*. IAEA-TECDOC 1529. © IAEA, 2007. Printed by the IAEA in Austria. February 2007. Wagramer Strasse 5, P.O. Box 100, A-1400 Vienna, Austria: International Atomic Energy Agency, 2007.

[30] *Boron 10*. [Accessed: 15-12-2023]. URL: `https://www.nuclear-power.com/glossary/boron-10/`.

[31] J. D. Feichtner, D. E. Thomas, and B. E. Yoldas. *Separation of isotopes of zirconium*. Tech. rep. 1986.

[32] *Basic Organization: MAT, MF MT*. [Accessed: 15-12-2023]. URL: `https://t2.lanl.gov/nis/endf/intro05.html`.

[33] *ADS Nuclear Data Library v2.0*. [Accessed: 15-12-2023]. URL: `https://www-nds.iaea.org/ads/adsgendf.html`.

[34] *ENDF MT Values*. [Accessed: 15-12-2023]. URL: `https://t2.lanl.gov/nis/endf/mts.html`.

[35] *ENDF/B-VI Incident-Neutron Data*. [Accessed: 15-12-2023]. URL: `https://t2.lanl.gov/nis/data/endf/endfvi-n.html`.

[36] International Association for the Properties of Water and Steam. *Revised Release on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam*. Tech. rep. IAPWS R7-97(2012). 2007.

[37] Robert Parrish. *getLebedevSphere*. [Accessed: 15-12-2023]. URL: `https://se.mathworks.com/matlabcentral/fileexchange/27097-getlebedevsphere`.

[38] V. I. Lebedev and D. N. Laikov. *A quadrature formula for the sphere of the 131st algebraic order of accuracy*. Tech. rep. 1999.

[39] *Iterative Methods for Linear and Nonlinear Equations: Matlab Codes*. [Accessed: 15-12-2023]. URL: `https://ctk.math.ncsu.edu/matlab_roots.html`.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Siim Erik Pugal

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Simplified Modelling of Neutron Transport in Python”, supervised by Marti Jeltsov
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

04.01.2024

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 - Addional Plots from the Monte Carlo Simulation

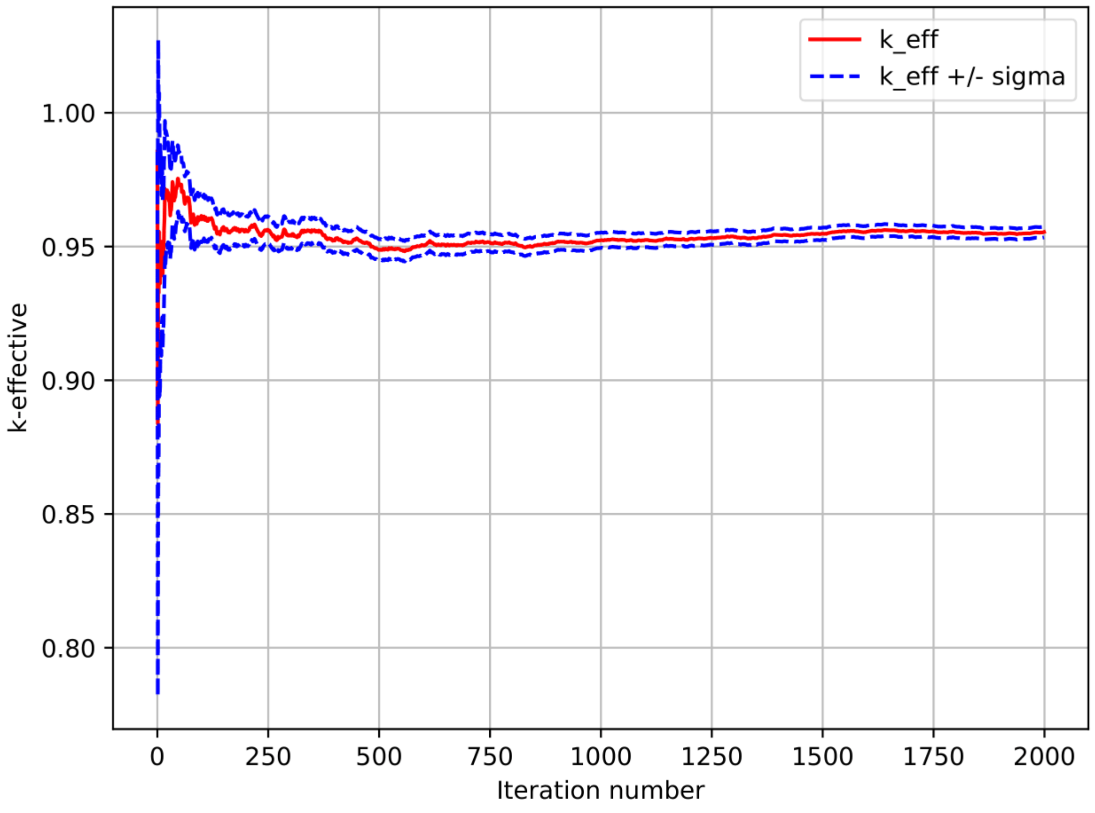


Figure 14. Graphical output of the k-effective value over the number of iteration cycles for the Monte Carlo method. In this case, the pitch value has be decreased from 3.6 cm to 2.6 cm, which results in a 10% drop in criticality.

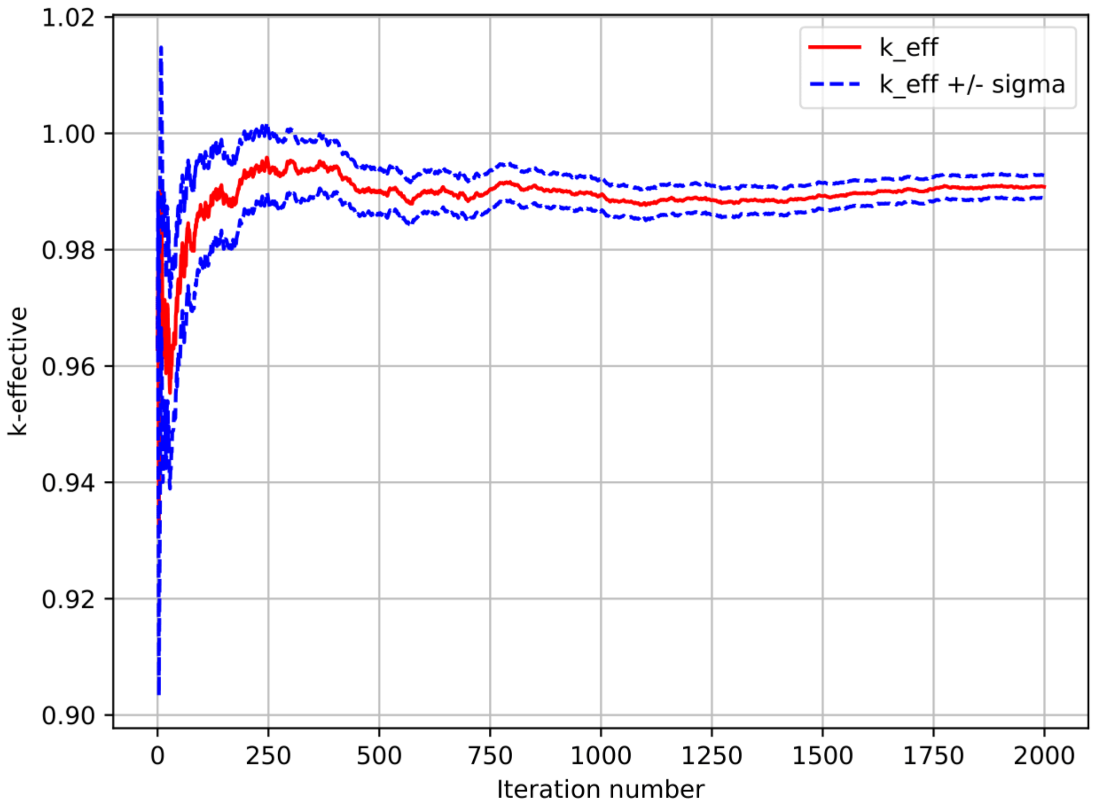


Figure 15. Graphical output of the k-effective value over the number of iteration cycles for the Monte Carlo method. In this case, the pitch still remains at 3.6 cm, but the unit cell walls have been increased from 0.2 cm to 0.4 cm. This lowers the criticality value around 6%.

Appendix 3 - Addional Plots from the Discrete Ordinates Simulation

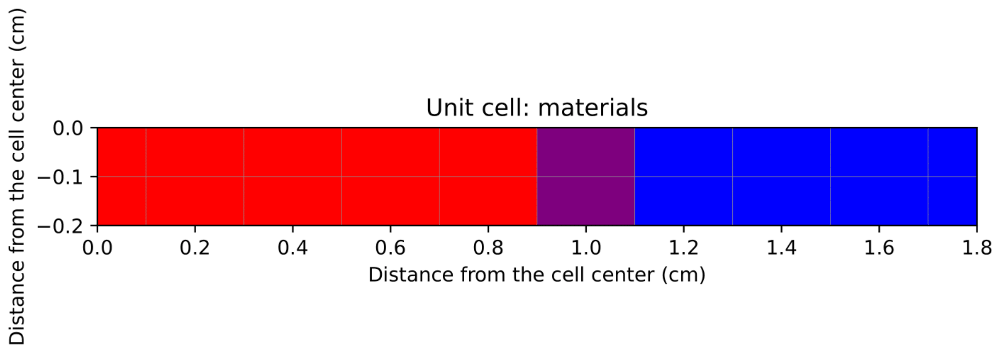


Figure 16. A 2D plot where different materials are visually represented by different colors or shades in the grid cells. It uses the material matrix to determine the material type at each grid location and assigns colors accordingly. The resulting plot provides a visual representation of how different materials are distributed within the unit cell.

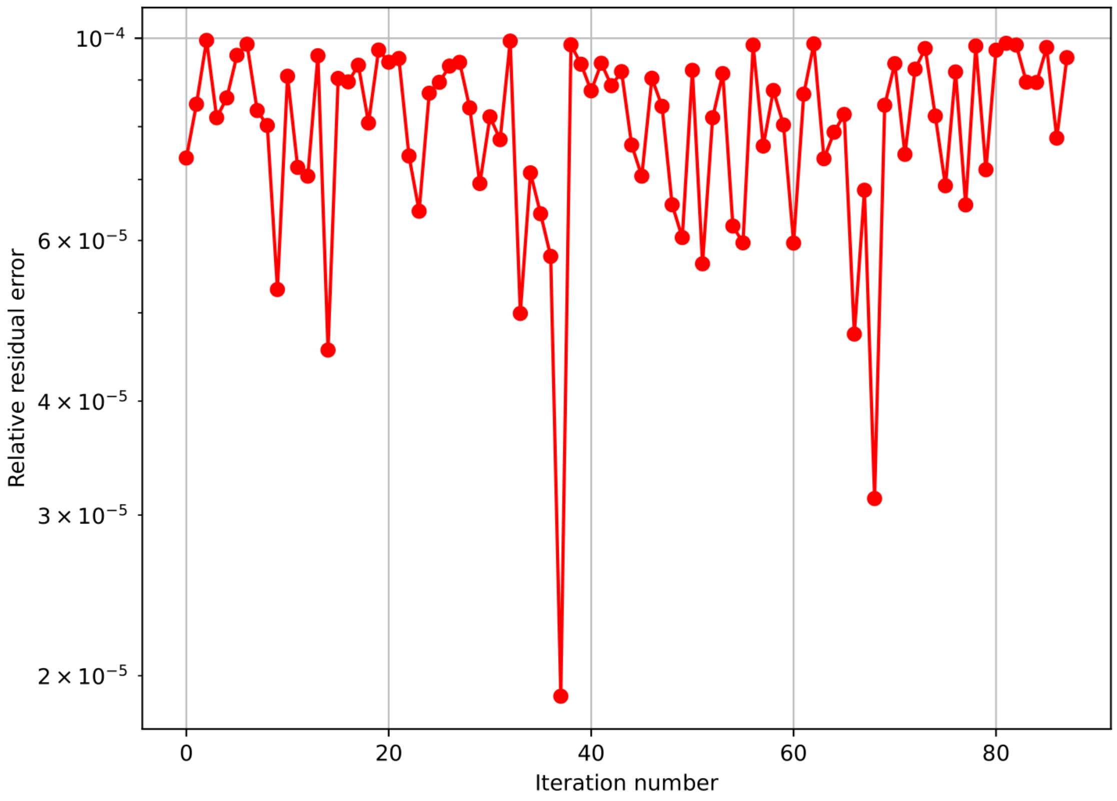


Figure 17. Plot of the residual errors during each iteration cycle obtained from the BiCGSTAB solver.

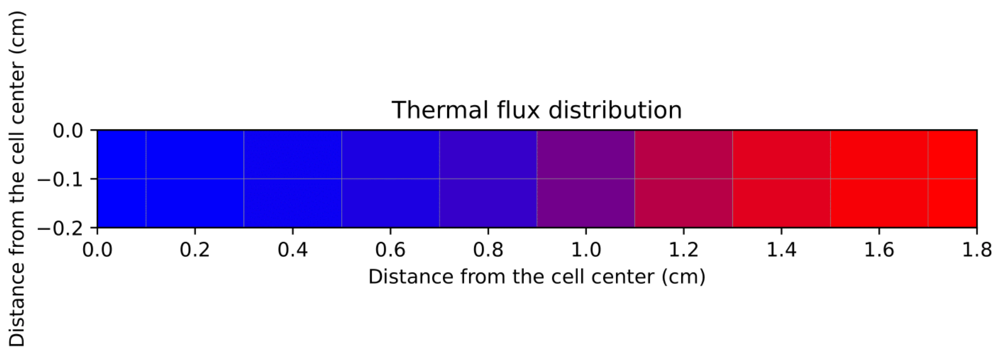


Figure 18. Scalar flux distribution in the thermal zone for each cell of the material matrix.

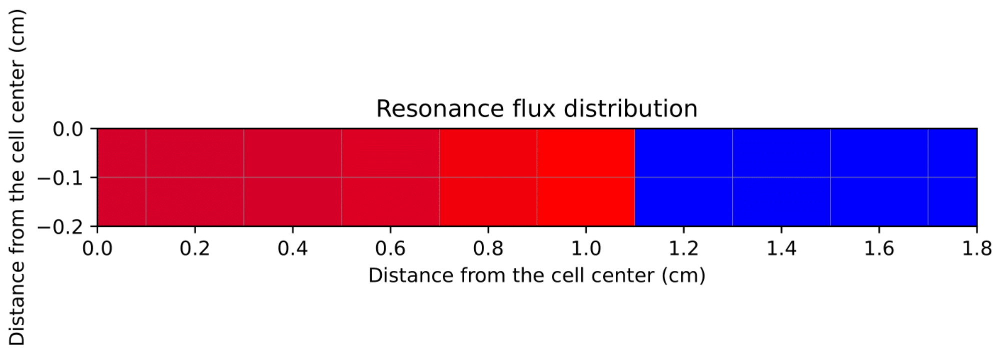


Figure 19. Scalar flux distribution in the resonance region for each cell of the material matrix.

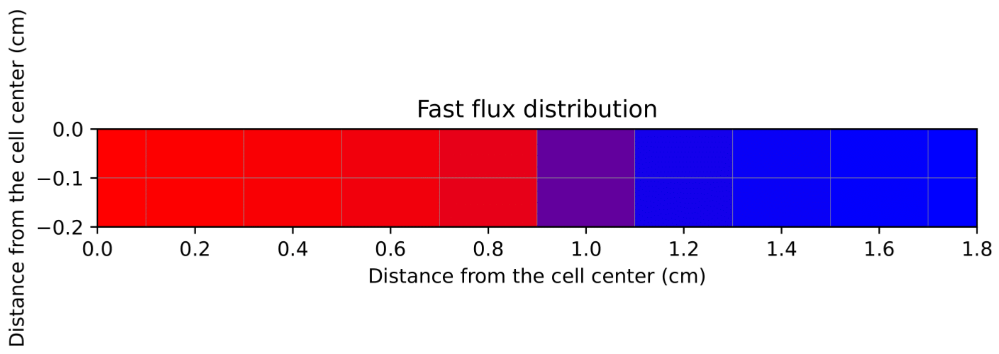


Figure 20. Scalar flux distribution in the fast zone for each cell of the material matrix.

Modified material matrix, which gave the graphical output represented in Figure 21.

```
# Define the material for each node
# (0 is coolant, 1 is cladding, 2 is fuel)
mat = np.array([[2, 2, 2, 1, 0, 0, 0, 0, 0, 0],
                [2, 2, 2, 1, 0, 0, 0, 0, 0, 0]])
```

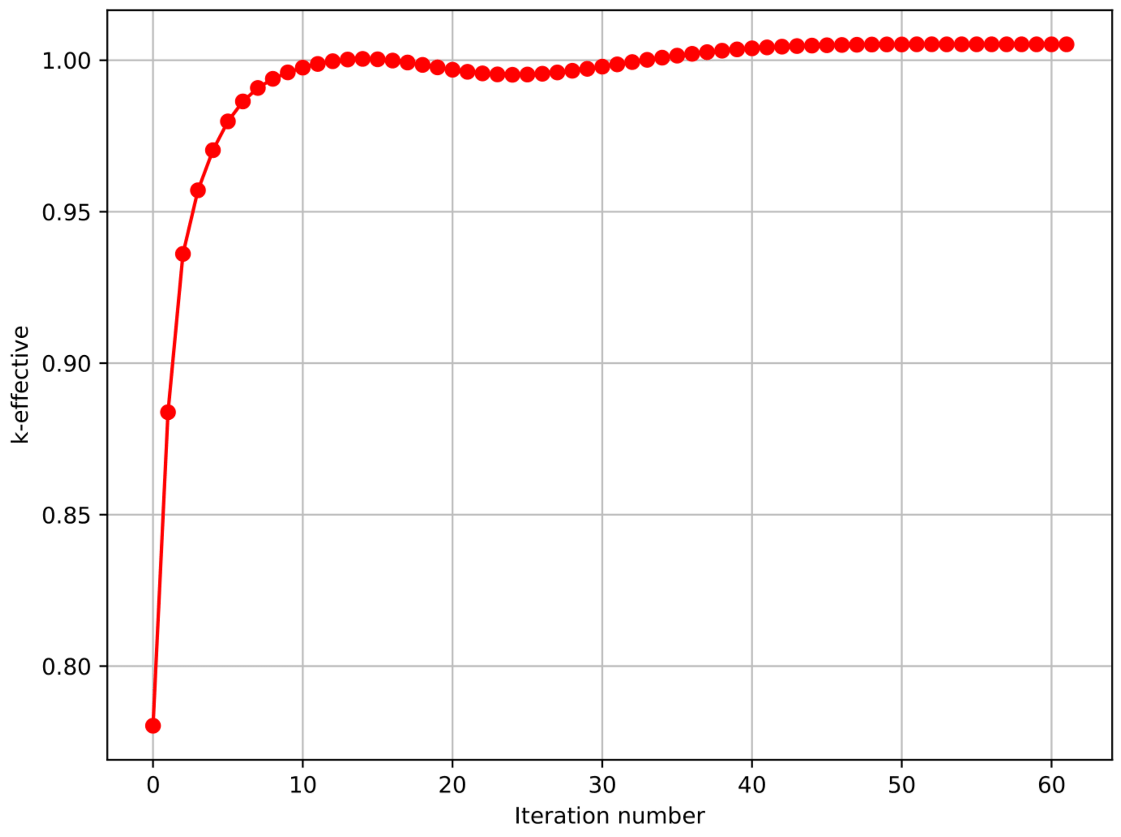


Figure 21. Alternative graphical output of the k-effective value over the number of iteration cycles for the discrete ordinates method. As descibed in the results chapter, the plot is the result of a modified material matrix with decreased fuel zones and increased coolant zones. This results in around a 4% decrease in criticality. Additionally, the convergence took less steps than in the original demo, which is a little over 60.

Appendix 4 – Source Codes

GitHub repository for all the source codes: `https://github.com/siimurik/BTE`