Development of a Python Module for a Neutron Transport Equation Solver using the Method of Characteristics in Two Dimensions

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science

in the
Department of Nuclear Engineering
Faculty of Engineering and Technology
University of Dhaka

Submitted by:

Exam Roll	:	3304
Registration No.	:	2013-712-220
Session	:	2017-2018

June, 2020

Certificate of Approval

The thesis titled "**Development of a Python Module for a Neutron Transport Equation Solver using the Method of Characteristics in Two Dimensions**" is done under my supervision, meets acceptable presentation standards and can be submitted for evaluation to the Department of Nuclear Engineering, University of Dhaka in partial fulfillment of the requirements for the degree of **Master of Science in Nuclear Engineering**.

Signature and Date

Dr. Khorshed Ahmad Kabir
Supervisor
UGC Professor
Department of Nuclear Engineering
Faculty of Engineering and Technology
University of Dhaka
Ramna, Dhaka.

Declaration

I hereby declare that the work presented in this thesis is the outcome of the investigations performed and authentically prepared by me under the supervision of **Dr. Khorshed Ahmad Kabir**, UGC Professor, Department of Nuclear Engineering, University of Dhaka. I also declare that no part of this thesis has been submitted elsewhere for the award of any degree or diploma.

Signature and Date

Kazi Azman Rafee

Abstract

A neutron transport solver was developed in Python by means of the Method of Characteristics (MOC) and was successfully deployed in solving one-group and multi-group eigenvalue problems in one dimension for both homogeneous and heterogeneous geometry and in two dimensions for heterogeneous rectangular geometries only. The solver was successfully parallelized with satisfactory performance metrics as observed in typical desktop computers. In assembly calculations, the solver performed well with uranium assemblies, but has some residual error in treating MOX assemblies, absorbing mediums and vacuum conditions, due to lack of adequate modeling of scattering anistropicity. In future, the solver's capabilities will be expanded in this regard by applying a higher order P_N approximation. The solvers applicability will be augmented so that it can handle unstructured geometries as well. The overall execution time will be minimized through incorporation of acceleration methods along with other optimization schemes.

Keywords: *Method of Characteristics(MOC), neutron transport solver, two dimensions, parallelization*

Acknowledgment

Being a novice in the world of scientific computation and programming in general, I must admit I felt very much intimidated by the whole idea of pursuing an undertaking of this level, and the sheer growth required of me to take on the hurdles it present. And it is these words, ”Everything seems impossible until it’s done”, that helped me find the courage to explore the unknown. From occasionally quoting Mandela for encouragement to analyzing situations with Murphy’s and Parkinson’s lenses, I often got astounded by his unparalleled wisdom and insight, without which this work would have never been possible. I will take this opportunity to express my bottomless gratitude to my supervisor, Dr. Khorshed Ahmad Kabir, for all that he provided me with without ever holding back, from ink and papers to words and wisdom and so much more.

I will also take this opportunity to thank the teachers in my department, home and abroad, for being supportive throughout and for advising me on various occasions. I thank my batch-mates for encouraging me from time to time in my pursuit of rather ambitious goals. I would also like to thank Syed Bahauddin Alam, for encouraging me into investing my efforts in developing MOC algorithm. I would have definitely opted for P_N method otherwise.

I thank my parents and my brother for all their love and kindness that allowed me to channel my focus into this project. I am also grateful to my friends for humoring me during the late night online gossips that helped me to rejuvenate from time to time, and for occasionally counseling me in resolve technical issues, which helped me in avoiding a lot of trouble to say the least.

And lastly, I would like to thank the Almighty, to grant me the luck to get this far. It has all been some happy accidents that allowed this venture to find success.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Because almost any nuclear technology frequently involves interaction with ionizing radiation, computer simulation is an absolute necessity to enable accurate prediction of the consequences beforehand. This is particularly a sensitive matter in case of nuclear power reactors, where human lives, national power grid and investment of billions of dollars are involved, and as such, its application is ever increasing to allow for better safety margin and for operation beyond design life [1]. Some key aspects of nuclear power operation that finds computer simulation necessary includes

- researching on nuclear safety,
- optimization of technical and economical parameters for power generation,
- designing reactor experiments,
- development of simulators to train operating personnels, etc. [2].

Codes are used to model neutronics (reactor physics), thermal-hydraulic behavior, material irradiation and so on in a nuclear reactor. The reactor physics codes are used to evaluate static designs, predict operational transients and analyze accidental situations. These codes are classified as cross section library processing codes, multigroup constant generation codes, static design codes, depletion codes, reactor kinetics codes and so on [3]. These design codes involve solving neutron transport equation or diffusion equation numerically.

The purpose of design codes is to find eigenvalue-k, also known as multiplication factor, and its eigenfunctions, i.e. neutron flux distributions. These outputs are used in determining excess reactivity, control rod worth, reactivity coefficient, power distributions and so on [4]. The design calculations are classified into two

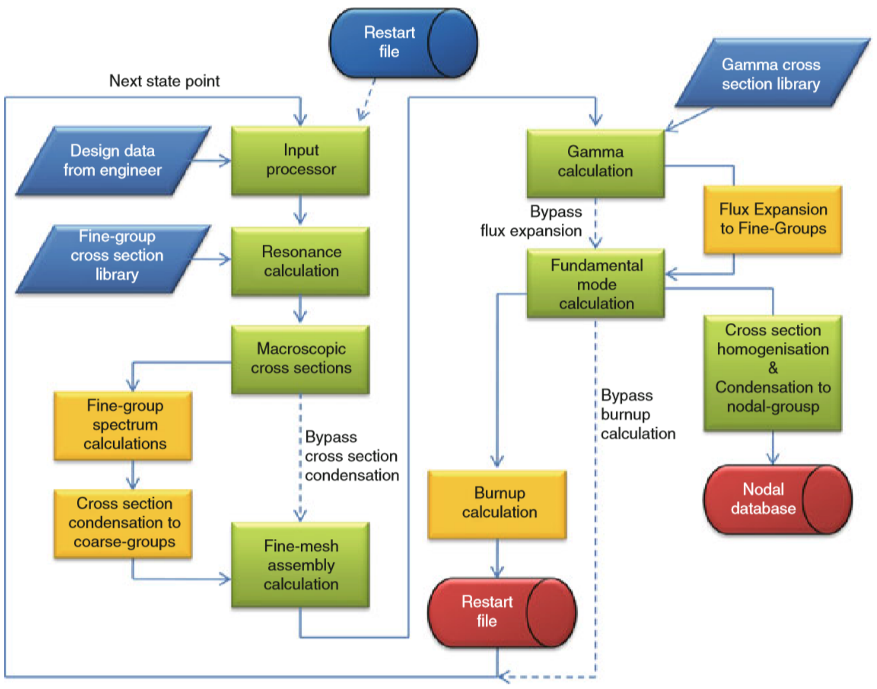

Restart file
Next state point
Gamma cross section library
Design data from engineer
Input processor
Gamma calculation
Flux Expansion to Fine-Groups
Bypass flux expansion
Fine-group cross section library
Resonance calculation
Fundamental mode calculation
Macroscopic cross sections
Cross section homogenisation & Condensation to nodal-group
Fine-group spectrum calculations
Bypass cross section condensation
Bypass burnup calculation
Burnup calculation
Nodal database
Cross section condensation to coarse-groups
Fine-mesh assembly calculation
Restart file

Figure 1.1: Typical Lattice Calculation Scheme [5]

categories: lattice physics calculations that provide group-constants for an assembly, and nodal calculations that use these group-constants to perform coupled neutronics-thermal hydraulics calculations through out the whole core [5]. Figure 1.1 summarizes the lattice calculation scheme.

Given the complexity of the geometry, energy dependency, angular dependency and the spatial dependency of neutron distributions, the overall calculation is split up into several stages to manage the scale of computations required. At first using a homogeneous or heterogeneous model, resonance calculations are performed using established mathematical routines. The continuous energy cross-sections gets discretized into ultra-fine group-constants. At this point, one dimensional calculations are performed on a pin-cell.

Using the results, the ultra-fine group-constants are collapsed to fine group-constants. With these fine group-constants, a two dimensional calculation is performed on an assembly. At this level, a suitable approximation is used to breakdown the angular dependency of neutron population. The fine group-constants get collapsed to few group-constants using the results of assembly calculations. These few group-constants are used to perform whole core calculations in three dimensions. At this level, diffusion approximation is usually used, thus the angular dependency is ignored altogether. Figure 1.2 summarizes the calculation flow in reactor design analysis.

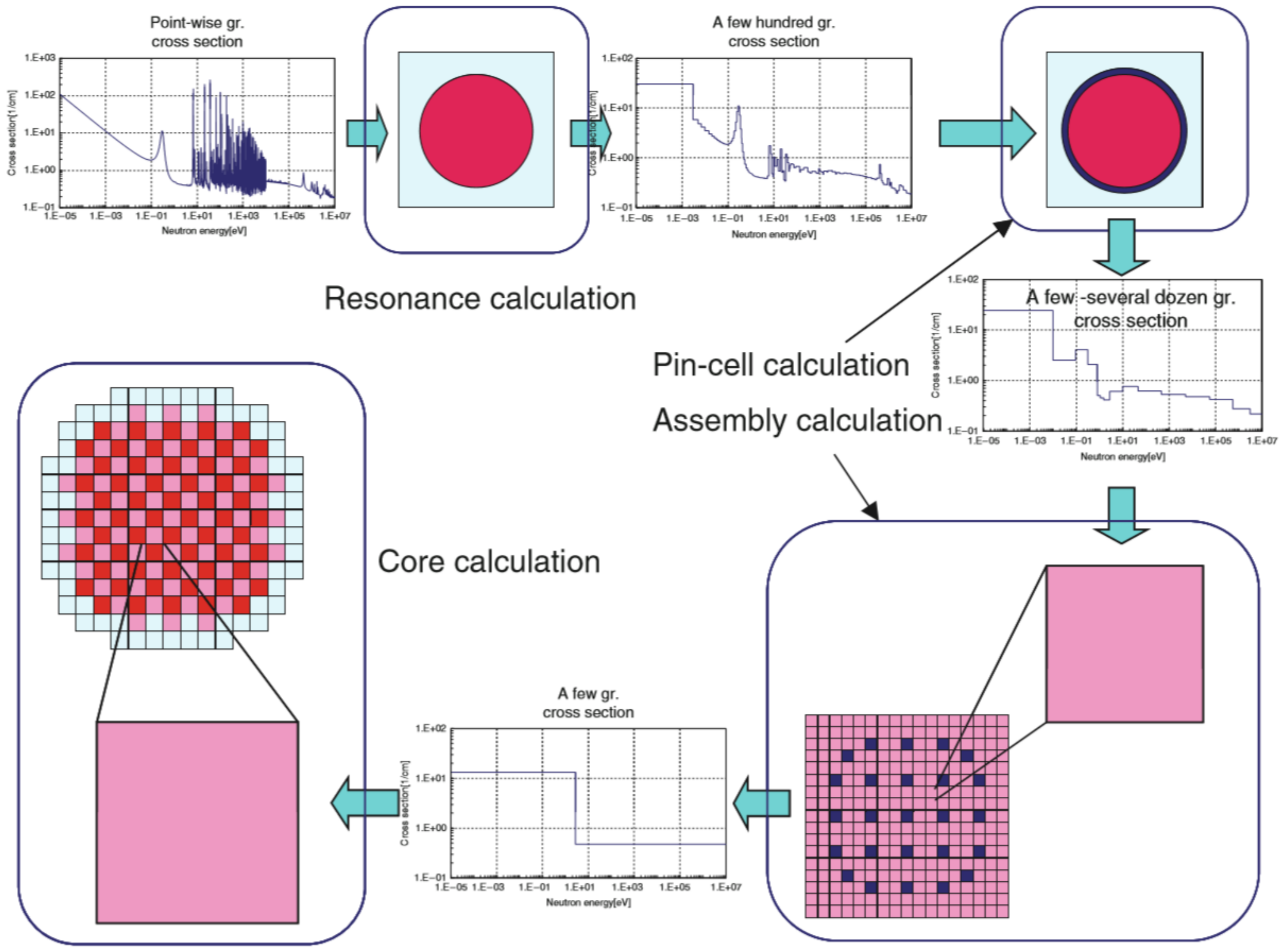


Figure 1.2: General Calculation Flow for Core Analysis [5]

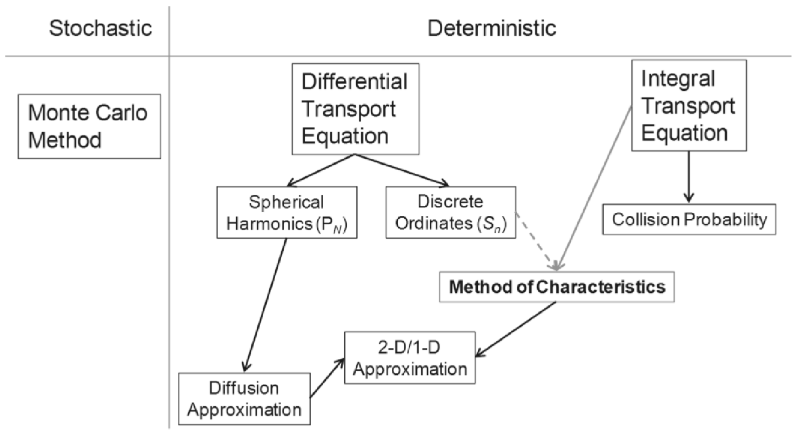

Stochastic
Deterministic
Monte Carlo Method
Differential Transport Equation
Integral Transport Equation
Spherical Harmonics (P_N)
Discrete Ordinates (S_n)
Collision Probability
Method of Characteristics
2-D/1-D Approximation
Diffusion Approximation

Figure 1.3: Taxonomy for Methods of solving Neutron Transport Equations [6]

Neutron transport equation, with some modifications, is solved to generate the desired results at each level. However, given the uniqueness in each case, the approach to solution varies accordingly. Usually, collision probability (CP) method is used for pin-cell, method of characteristics (MOC) is used for assembly and nodal method is used for whole-core calculations. There are a host of other methods for solving neutron transport as well, such as discrete-ordinates (S_N) method that considers angular flux at N specific directions to account for angular dependency, spherical harmonics (P_N) method that applies a series approximation of N terms to account for angular dependency, Monte Carlo method and so on. Figure 1.3 provides a brief summary of these methods.

Though Monte Carlo (MC) method is the most accurate and versatile, it has a very high computational requirement to achieve the desired accuracy, and so it is usually used as a precision code. On the other hand, deterministic methods are usually applicable for regular geometry and finds a narrow scope of applications. They are tailored to solve power reactor problems and is much faster than Monte Carlo, though less accurate (but at an acceptable level). These methods are adopted in engineering codes and is used in simulating reactor campaigns by operating personnel [2].

The P_N approximation or S_N approximation can be used to solve problems of any scale in theory by varying "N", but for practical purposes, their applications are limited to core or assembly calculations. At these scales, the most versatile method for providing transport solution is method of characteristics (MOC).

MOC is based on the integral formulation of neutron transport as derived in Subsection 2.2 on page 21. Rather than any complex linear algebra routine, it uses ray-tracing which is simpler to implement independent of number of energy groups. It allows flexibility of handling angular dependency as required of the geometry and also allows easy handling of upscattering.

It was initially unpopular due to memory storage requirement but with progress in computational capability, it has now become popular and most preferable for assembly calculations, and is used for whole core calculations as well. It can be customized to handle unstructured geometries, scattering anisotropicity, high flux gradient and so on, and thus, it is the most versatile deterministic method for solving neutron transport problems. Extensive literature on this method is given by Askew [7], Halsall [8], Goldgerg et al. [9], Cho and Hong [10], Roy [11], Postma and Vujic [12], Kosaka and Saji [13], Kim et al. [14], Jevremovic et al. [15], Hébert [16], Knott and Yamamoto [5], etc.

Application of MOC was first proposed by Askew [7] in 1972 and was added in WIMS as CACTUS module [8] in 1980. However, it was not until 1990s the method became popular, and was incorporated in lattice physics codes such as Dragon [17], CASMO-4 [18], AEGIS [19], LANCER02 [20], HELIOS-2 [21], APOLLO-2 [22] and most recently in Laputa [23] by Tsinghua University. It is also incorporated in codes for High Performance Computing systems for 3D whole-core calculations such as DeCART [24], OpenMOC [25], MPACT [26, 27], etc. and for transient analysis [28] as well.

Bangladesh has just embarked on its nuclear power program. To maneuver it successfully in the long run, the country has to develop its indigenous capability for handling nuclear technology. This will require progress at many frontiers, including computational capabilities. Almost all countries with national nuclear power program have their own computer simulation tools developed locally. As such, Bangladesh needs to start developing these simulation tools that will help internalize the nuclear technology in coming decades and foster research in this arena nationwide.

Developing indigenous computational tools will require a lot of investment, time and human effort, and as such, the author aspires to provide motivation for such intentions through his M. Sc. project. Implementation of MOC in 2D geometry seemed a good starting point for developing a lattice physics code.

Easier said than done, the MOC algorithm used in expensive high quality lattice codes is not a realistic goal given the time frame of the project and the effort needed. So the objective was set to a simplistic vanilla implementation in structured rectangular geometry using rectangular meshes. The algorithm was developed in Python 3 using the IDE of Spyder on Windows. Python is an interpretative language which allows fast development of any algorithm for scientific computation and provides ease of data handling, though it is quite slower than C++. One way to offset such disadvantage is to parallelize the code which is quite easier with Python multiprocessing module, and so the goal was set to develop a parallelized MOC algorithm.

However, writing a parallelized multigroup 2D MOC solver from scratch is a big leap to take. To proceed step by step, at first, a one-group 1D solver was developed to test the algorithm on homogeneous and heterogeneous geometry, optimizing the algorithm and probing its sensitivity along the way. It was then modified and a multigroup solver was developed. The iterative scheme was optimized at this stage to make it as fast as possible. The parallelization scheme was also designed at this stage. Successful implementation of the algorithm in 1D provided much needed learning experiences and confidence to move on to the next step.

The one-group 2D Solver was developed and tested on two benchmark problems. The solver was then modified for multigroup problems and parallelized with the scheme designed beforehand. The multigroup solver was tested on different benchmark problems. All the results are reported in Chapter 4 in a progressive manner.

In Chapter 2, the underlying theories and principles of the neutron transport equation and the method of characteristics as a PDE solver is discussed. A brief description of Python language is also provided. In Chapter 3, the approximations used in modifying the neutron transport is discussed in detail, followed by discussions on the iterative scheme used. Afterwards, the implementation of MOC algortihm both in one dimension and two dimensions is also explained in detail, justifying the author's decisions. The chapter concludes with details of the parallelization scheme adopted.

After reporting the results with discussions, the author finally draws conclusion in Chapter 5. A list of the sources cited is then provided followed by appendices at the end.

Chapter 2

Theory

2.1 Theory of Neutron Transport

2.1.1 Introductory Concepts

The discussions that follow here on are mostly inspired by Bell and Glasstone [29]. Let us define the following terms that characterize the neutron population in the context of reactor physics.

Neutron angular density or just *angular density*, denoted by $n(\vec{r}, \hat{\Omega}, E, t)$, is the expected number of neutrons at position $\vec{r}$ in direction $\hat{\Omega}$ with energy E at time t. Hence,

$$\begin{array}{c}\text{the number of neutrons in a volume } d^3r \\ \text{about } \vec{r} \text{ within a solid angle of } d\hat{\Omega} \text{ about} \\ \hat{\Omega} \text{ and energies in } dE \text{ about } E\end{array} = n(\vec{r}, \hat{\Omega}, E, t)\, d^3r\, d\hat{\Omega}\, dE. \tag{2.1}$$

Consequently, **neutron angular flux** or just *angular flux*, denoted by $\phi(\vec{r}, \hat{\Omega}, E, t)$, is the product of *angular density* n and speed of the neutron v. Hence we can write

$$\phi(\vec{r}, \hat{\Omega}, E, t) = vn(\vec{r}, \hat{\Omega}, E, t). \tag{2.2}$$

We can also define another quantity called **neutron vector flux** or just *vector flux*, denoted by $\vec{\phi}(\vec{r}, \hat{\Omega}, E, t)$, which is the product of the velocity $\vec{v}$ of the neutron and its *angular density*. Hence,

$$\vec{\phi}(\vec{r}, \hat{\Omega}, E, t) = \vec{v}n(\vec{r}, \hat{\Omega}, E, t). \tag{2.3}$$

Using the above definitions, we can define **neutron density** denoted by $N(\vec{r}, E, t)$, **total flux** or just *flux* denoted by $\Phi(\vec{r}, E, t)$, and **neutron current** denoted by

$\vec{J}(\vec{r}, E, t)$, as the integral of *angular density*, *angular flux* and *vector flux* over all directions, i.e.

$$N(\vec{r}, E, t) = \oiint_{4\pi} d\hat{\Omega} \;\; n(\vec{r}, \hat{\Omega}, E, t), \tag{2.4}$$

$$\Phi(\vec{r}, E, t) = \oiint_{4\pi} d\hat{\Omega} \;\; \phi(\vec{r}, \hat{\Omega}, E, t), \tag{2.5}$$

and

$$\vec{J}(\vec{r}, E, t) = \oiint_{4\pi} d\hat{\Omega} \;\; \vec{\phi}(\vec{r}, \hat{\Omega}, E, t). \tag{2.6}$$

We also define an additional term called **fluence**, $\mathit{\Phi}_T$, to describe neutron population. It is simply the integral of *flux* over a period of time T.

$$\mathit{\Phi}_T(\vec{r}, E) = \int_T dt \;\; \Phi(\vec{r}, E, t). \tag{2.7}$$

The free moving neutrons collide and thus interact with the nuclei of the atoms present in the system. The interaction can be a scattering, a radiative capture, a knockout or a fission. The probability of such interactions occurring are quantitatively described as cross sections.

Let us consider an uniform mono-energetic beam of projectiles, which are neutrons in this scenario, hitting perpendicularly a thin slab of target with the total number of target nuclei present equals X occupying a volume of V. For interaction i, the **interaction rate** R_i will be proportional to *flux* Φ and X, i.e.

$$R_i \propto \Phi X$$

or,

$$R_i = \sigma_i \Phi X.$$

Here σ_i is the proportionality constant and is defined as the **microscopic cross section** for the interaction i.

The **interaction rate density** for interaction i, $\bar{R}_i$ will be the ratio of R_i and V.

$$\bar{R}_i = \frac{\sigma_i \Phi X}{V}.$$

The ratio of X and V is the **number density** of target nuclei denoted by $\bar{X}$. Hence, we can write

$$\bar{R}_i = \sigma_i \Phi \bar{X}.$$

The quantities σ_i and $\bar{X}$ both depend on the medium. It is convenient to address their product as a single quantity called **macroscopic cross section**, Σ_i. Thus, the *interaction rate density* can be simply expressed as

$$\bar{R}_i = \Sigma_i \Phi. \tag{2.8}$$

The distinction in naming σ and Σ comes from recognizing that qualitatively, the *microscopic cross section* is simply the probability that a single nuclei interacts in the fluence of a single neutron per unit area, where as the *macroscopic cross section* is the expected number of interactions per unit volume of the medium for the same fluence, which can also be phrased as the probability of interaction when a neutron travels an unit distance.

So far we have discussed interactions in the context of an uniform mono-energetic neutron beam. However, in a reactor or other applications, the neutrons are not mono-energetic but rather form a continuous spectrum. The cross sections are strongly dependent on the neutron energy. As such, the reaction rate density $\bar{R}_i$ for interaction i at location $\vec{r}$ will be the total of all reaction rate densities at individual energies. For neutrons within energies dE about E, the interaction rate density $d\bar{R}_i$ is

$$d\bar{R}_i = \Sigma_i(\vec{r}, E)\Phi(\vec{r}, E, t)\, dE. \tag{2.9}$$

Hence, for a neutron spectrum in the energy-width E_g, the total reaction rate density $\bar{R}_i^g$ is

$$\bar{R}_i^g = \int_{E_g} dE \;\; \Sigma_i(\vec{r}, E)\Phi(\vec{r}, E, t). \tag{2.10}$$

Thus, for a medium in region C of volume V_c, the overall reaction rate $R_i^{g,c}$ for a spectrum in energy-width E_g is

$$R_i^{g,c} = \int_{V_c} d^3r \int_{E_g} dE \;\; \Sigma_i(\vec{r}, E)\Phi(\vec{r}, E). \tag{2.11}$$

In deterministic approaches to solving static neutronics problems numerically, it is impractical to use the continuous function of cross section against neutron energy, also referred to as excitation function, $\sigma(E)$, directly due to the insurmountable computational effort required. The usual practice is to discretize it by energy groups appropriately based on considerations such as even lethargy change, resonance peaks, upscattering region and so on, and then to calculate an average value of the cross section, referred to as **cell-averaged group-constant** or just *group-constant*, for the discrete energy group in a fixed region, identified as a cell.

The volume-averaged flux $\Phi_{g,c}$ for a certain energy group in the cell C, within the energy width E_g, is

$$\Phi_{g,c} = \frac{\displaystyle\int_{V_c} d^3r \int_{E_g} dE \;\; \Phi(\vec{r}, E)}{V_c}. \tag{2.12}$$

Drawing parallels with the reaction rate for mono-energetic neutrons in equation (2.8), we can write

$$R_i^{g,c} = \Sigma_i^{g,c} \Phi_{g,c} V_c \tag{2.13}$$

where $\Sigma_i^{g,c}$ is the group-constant of the cell for interaction i in energy-width E_g. Thus,

$$\Sigma_i^{g,c} = \frac{\displaystyle\int_{V_c} d^3r \int_{E_g} dE \;\; \Sigma_i(\vec{r}, E)\Phi(\vec{r}, E)}{\displaystyle\int_{V_c} d^3r \int_{E_g} dE \quad \Phi(\vec{r}, E)}. \tag{2.14}$$

Neutron sources in transport calculations are generally categorized into *independent sources* and *dependent sources*. Independent sources do not depend on the neutron density at the location of origin, for example (α, n) reactions, spontaneous fission reactions, etc. The independent sources are collectively represented by $Q_{ind}(\vec{r}, \hat{\Omega}, E, t)$, which is the probability that a neutron of energy E moving in direction $\hat{\Omega}$ originates in location $\vec{r}$ at time t in an unit time from independent sources. This means that, in a differential volume d^3r at location $\vec{r}$, the expected number of neutrons generated from independent sources with energies within dE about E moving in a direction within $d\hat{\Omega}$ about $\hat{\Omega}$ within a duration of dt about t is

$$Q_{ind}(\vec{r}, \hat{\Omega}, E, t)\, d^3r\, d\hat{\Omega}\, dE\, dt.$$

Dependent sources are those in which neutrons generated in a specific phase-space depends on the neutron angular density distribution at the location of origin. These neutrons typically originates from interactions leading to scattering and fission, but can also result from $(n, 2n)$ and other knock-out processes. To account for neutrons emerging from such processes, a **differential cross section**, Σ_i', is defined as the product of $\Sigma_i(\vec{r}, E)$, the macroscopic cross section for interaction i, and its **transfer probability distribution**, $f_i(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)$.

$$\Sigma_i' = \Sigma_i(\vec{r}, E) f_i(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E). \tag{2.15}$$

Here f_i is a function such that

$$f_i(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)\, d\hat{\Omega}\, dE$$

is the probability a neutron of energy within dE of E moving in a direction within $d\hat{\Omega}$ about $\hat{\Omega}$ emerges when a neutron of energy E' moving towards $\hat{\Omega}'$ undergoes an interaction of type i at location $\vec{r}$. The transfer probability distributions are normalized so that

$$\oiint_{4\pi} d\hat{\Omega} \int_0^\infty dE \;\; f_i(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)$$

exactly equals the number of neutrons that emerge from the interaction. For scattering, the above integral equals 1. For $(n, 2n)$ reactions, the integral is equal to 2. For fission, the integral is equal to $\bar{\nu}(\vec{r}, E')$, which is the average number of neutrons produced by a fission at $\vec{r}$ induced by a neutron of energy E'.

The total probability of neutron transfer per unit distance from $(\hat{\Omega}', E')$ to $(\hat{\Omega}, E)$, denoted by Σ_t', is the summation of Σ_i' for all interaction i. Thus we can write

$$\begin{aligned}\Sigma_t' &= \sum_i \Sigma_i' \\ &= \sum_i \Sigma_i(\vec{r}, E') f_i(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E).\end{aligned}$$

If $\Sigma_t(\vec{r}, E)$ is the total macroscopic cross section for all interactions i, i.e.

$$\Sigma_t(\vec{r}, E) = \sum_i \Sigma_i(\vec{r}, E),$$

we can write

$$\Sigma_t' = \;\Sigma_t(\vec{r}, E) f(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E), \tag{2.16}$$

where $f(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)$ is the overall transfer probability distribution at $\vec{r}$.

We can find the probability per unit time at which a neutron emerges from dependent sources at $\vec{r}$ with energy E moving towards $\hat{\Omega}$, from the angular flux $\phi(\vec{r}, \hat{\Omega}', E', t)$ by multiplying it with Σ_t'. Hence, the probability that a neutron of energy E moving in direction $\hat{\Omega}$ originates in location $\vec{r}$ at time t in an unit time, denoted by $Q_{dep}(\vec{r}, \hat{\Omega}, E, t)$, is

$$Q_{dep}(\vec{r}, \hat{\Omega}, E, t) = \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_t' \, \phi(\vec{r}, \hat{\Omega}', E', t).$$

Substituting Σ_t' using equation (2.16), we get

$$Q_{dep}(\vec{r}, \hat{\Omega}, E, t) = \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_t(\vec{r}, E') f(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E) \, \phi(\vec{r}, \hat{\Omega}', E', t). \tag{2.17}$$

2.1.2 Derivation of Neutron Transport Equation

Let us consider a packet of neutrons which at time t is existing in a volume of d^3r about $\vec{r}$ having energies within range dE about E moving in directions within $d\hat{\Omega}$ about $\hat{\Omega}$. So,

$$\begin{matrix}\text{Number of neutrons}\\ \text{initially in the packet}\end{matrix} = n(\vec{r},\hat{\Omega},E,t)\,d^3r\,d\hat{\Omega}\,dE \tag{2.18}$$

as per equation (2.1).

After a small time period of Δt, the packet will be displaced by $v\Delta t$ in the direction $\hat{\Omega}$, where v is the speed of the packet. Hence, its new location will be $\vec{r}+v\Delta t\,\hat{\Omega}$.

Neutrons will collide or interact with nuclei and thus will get lost from the packet as they travel during Δt. Similar to equation (2.9),

$$\begin{matrix}\text{Number of neutrons lost from}\\ \text{packet due to interaction during } \Delta t\end{matrix} = \Sigma_t(\vec{r},E)\phi(\vec{r},\hat{\Omega},E,t)\,\Delta t\;d^3r\,d\hat{\Omega}\,dE \tag{2.19}$$

where Σ_t is the total macroscopic cross-section as discussed before. Hence, using equations (2.18) and (2.19), we find

$$\begin{aligned}\begin{matrix}\text{Number of initial}\\ \text{neutrons surviving in}\\ \text{the packet after } \Delta t\end{matrix} &= \begin{matrix}\text{Number of neutrons}\\ \text{initially in the packet}\end{matrix} - \begin{matrix}\text{Number of neutrons lost}\\ \text{from packet due to}\\ \text{interaction during } \Delta t\end{matrix}\\ &= n(\vec{r},\hat{\Omega},E,t)\,d^3r\,d\hat{\Omega}\,dE\\ &\qquad - \Sigma_t(\vec{r},E)\phi(\vec{r},\hat{\Omega},E,t)\,\Delta t\;d^3r\,d\hat{\Omega}\,dE\\ &= \{n(\vec{r},\hat{\Omega},E,t) - \Sigma_t(\vec{r},E)\phi(\vec{r},\hat{\Omega},E,t)\,\Delta t\}\,d^3r\,d\hat{\Omega}\,dE.\end{aligned}$$

Substituting $\phi(\vec{r},\hat{\Omega},E,t)$ from equation (2.2) in the above equation and rearranging, we get

$$\begin{matrix}\text{Number of neutrons lost}\\ \text{from packet due to}\\ \text{interaction during } \Delta t\end{matrix} = n(\vec{r},\hat{\Omega},E,t)\,\{1 - v\Delta t\,\Sigma_t(\vec{r},E)\}\,d^3r\,d\hat{\Omega}\,dE. \tag{2.20}$$

As the packet travels, the neutrons generated at location $\vec{r}$, both dependently and independently, enters the packet.

$$\begin{array}{c}\text{Number of neutrons}\\ \text{generated at } \vec{r}\\ \text{that entered the packet}\end{array} = \{Q_{dep}(\vec{r},\hat{\Omega},E,t)+Q_{ind}(\vec{r},\hat{\Omega},E,t)\}\,\Delta t\;\; d^3r\,d\hat{\Omega}\,dE. \tag{2.21}$$

Applying conservation principle,

$$\begin{array}{c}\text{Number of neutrons}\\ \text{in a packet at}\\ \text{location } \vec{r}+v\Delta t\,\hat{\Omega}\end{array} = \begin{array}{c}\text{Number of initial neutrons}\\ \text{surviving in the}\\ \text{packet after } \Delta t\end{array} + \begin{array}{c}\text{Number of neutrons}\\ \text{generated at } \vec{r} \text{ that}\\ \text{entered the packet.}\end{array}$$

Substituting equations (2.20) and (2.21) in the above equation and dividing by $d^3r\,d\hat{\Omega}\,dE$, we get

$$\begin{aligned} n(\vec{r}+v\Delta t\,\hat{\Omega},\hat{\Omega},E,t+\Delta t) &= n(\vec{r},\hat{\Omega},E,t)\,\{1-v\Delta t\,\Sigma_t(\vec{r},E)\}\\ &\quad+\{Q_{dep}(\vec{r},\hat{\Omega},E,t)+Q_{ind}(\vec{r},\hat{\Omega},E,t)\}\,\Delta t. \end{aligned} \tag{2.22}$$

Rearranging, we get

$$\begin{aligned} &\frac{n(\vec{r}+v\Delta t\,\hat{\Omega},\hat{\Omega},E,t+\Delta t)-n(\vec{r},\hat{\Omega},E,t)}{\Delta t}\\ &\qquad+v\,\Sigma_t(\vec{r},E)\,n(\vec{r},\hat{\Omega},E,t)\\ &\qquad\qquad= Q_{dep}(\vec{r},\hat{\Omega},E,t)+Q_{ind}(\vec{r},\hat{\Omega},E,t). \end{aligned} \tag{2.23}$$

Taking $\lim_{\Delta t\to 0}$, the first term in the left-hand side of equation (2.23),

$$\begin{aligned} &\lim_{\Delta t\to 0}\left[\frac{n(\vec{r}+v\Delta t\,\hat{\Omega},\hat{\Omega},E,t+\Delta t)-n(\vec{r},\hat{\Omega},E,t)}{\Delta t}\right]\\ &\qquad=\lim_{\Delta t\to 0}\left[\frac{n(\vec{r},\hat{\Omega},\hat{\Omega},E,t+\Delta t)-n(\vec{r},\hat{\Omega},E,t)}{\Delta t}\right]\\ &\qquad+\lim_{\Delta t\to 0}\left[\frac{n(\vec{r}+v\Delta t\,\hat{\Omega},\hat{\Omega},E,t+\Delta t)-n(\vec{r},\hat{\Omega},E,t+\Delta t)}{\Delta t}\right]. \end{aligned} \tag{2.24}$$

The first limit on the right-hand side,

$$\lim_{\Delta t\to 0}\left[\frac{n(\vec{r},\hat{\Omega},\hat{\Omega},E,t+\Delta t)-n(\vec{r},\hat{\Omega},E,t)}{\Delta t}\right]=\frac{\partial}{\partial t}n(\vec{r},\hat{\Omega},E,t). \tag{2.25}$$

Substituting $\Delta r = v\Delta t$, the second limit on the right-hand side,

$$\begin{aligned} &\lim_{\Delta t\to 0}\left[\frac{n(\vec{r}+\Delta r\,\hat{\Omega},\hat{\Omega},E,t+\Delta t)-n(\vec{r},\hat{\Omega},E,t+\Delta t)}{\Delta t}\right]\\ &=\lim_{\Delta r\to 0}\left[\frac{n(\vec{r}+\Delta r\,\hat{\Omega},\hat{\Omega},E,t+\Delta t)-n(\vec{r},\hat{\Omega},E,t+\Delta t)}{\Delta r}\right]\times\lim_{\Delta t\to 0}\left[\frac{\Delta r}{\Delta t}\right]. \end{aligned} \tag{2.26}$$

Now,

$$\lim_{\Delta t \to 0} \left[\frac{\Delta r}{\Delta t} \right] = v. \tag{2.27}$$

Using equation (2.27) and equation (2.2) in equation (2.26), we get

$$\lim_{\Delta t \to 0} \left[\frac{n(\vec{r} + \Delta r\, \hat{\Omega}, \hat{\Omega}, E, t + \Delta t) - n(\vec{r}, \hat{\Omega}, E, t + \Delta t)}{\Delta t} \right]$$
$$= \lim_{\Delta r \to 0} \left[\frac{\phi(\vec{r} + \Delta r\, \hat{\Omega}, \hat{\Omega}, E, t + \Delta t) - \phi(\vec{r}, \hat{\Omega}, E, t + \Delta t)}{\Delta r} \right]. \tag{2.28}$$

Now, using the fact that very small change in flux can be written as dot-product of gradient of flux and change in position vector, and dropping Δt, we get

$$\lim_{\Delta r \to 0} \left[\frac{\phi(\vec{r} + \Delta r\, \hat{\Omega}, \hat{\Omega}, E, t) - \phi(\vec{r}, \hat{\Omega}, E, t)}{\Delta r} \right] = \frac{\nabla \phi(\vec{r}, \hat{\Omega}, E, t) \cdot (dr\, \hat{\Omega})}{dr}$$
$$= \hat{\Omega} \cdot \nabla \phi(\vec{r}, \hat{\Omega}, E, t). \tag{2.29}$$

Using equation (2.25), equation (2.28) and equation (2.29) in equation (2.24), we get

$$\lim_{\Delta t \to 0} \left[\frac{n(\vec{r} + v\Delta t\, \hat{\Omega}, \hat{\Omega}, E, t + \Delta t) - n(\vec{r}, \hat{\Omega}, E, t)}{\Delta t} \right]$$
$$= \frac{\partial}{\partial t} n(\vec{r}, \hat{\Omega}, E, t) + \hat{\Omega} \cdot \nabla \phi(\vec{r}, \hat{\Omega}, E, t). \tag{2.30}$$

Substituting the first term on the left-hand side in equation (2.23) from equation (2.30), we get

$$\frac{\partial}{\partial t} n(\vec{r}, \hat{\Omega}, E, t) + \hat{\Omega} \cdot \nabla \phi(\vec{r}, \hat{\Omega}, E, t) + v\, \Sigma_t(\vec{r}, E)\, n(\vec{r}, \hat{\Omega}, E, t)$$
$$= Q_{dep}(\vec{r}, \hat{\Omega}, E, t) + Q_{ind}(\vec{r}, \hat{\Omega}, E, t).$$

Finally, substituting $n(\vec{r}, \hat{\Omega}, E, t)$ from equation (2.2) and $Q_{dep}(\vec{r}, \hat{\Omega}, E, t)$ from (2.17) in the above equation, we get

$$\frac{1}{v} \frac{\partial}{\partial t} \phi(\vec{r}, \hat{\Omega}, E, t) + \hat{\Omega} \cdot \nabla \phi(\vec{r}, \hat{\Omega}, E, t) + \Sigma_t(\vec{r}, E)\, \phi(\vec{r}, \hat{\Omega}, E, t)$$
$$= \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_t(\vec{r}, E') f(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)\, \phi(\vec{r}, \hat{\Omega}', E', t)$$
$$+ Q_{ind}(\vec{r}, \hat{\Omega}, E, t). \tag{2.31}$$

Equation (2.31) is known as the **Boltzmann Neutron Transport Equation**, or more commonly as *Neutron Transport Equation.* It is hardly possible to solve this equation as it is. In the coming sections and in chapter 3, some of the modifications that are applied to solve this equation will be discussed.

2.1.3 Interface and Boundary Conditions

In the derivation of equation (2.31), it was assumed that the cross sections are a continuous functions of $\vec{r}$ in its vicinity. However, in practice, the transport equation is solved for heterogeneous geometries with interfaces between different mediums. The cross sections are thus, no longer a continuous function of $\vec{r}$ which necessitates mathematical considerations in solving transport problem.

When a packet of neutron packet just crosses a physical interface, there is no reason that the number of neutrons in the packet will change. Hence, if s is the distance traveled by the packet along $\hat{\Omega}$, then $N(\vec{r}+s\hat{\Omega},\hat{\Omega},E,t+\frac{s}{v})$ will be a continuous function of s. So, neutron angular density and angular flux, as well as neutron density and total flux, does not change abruptly in crossing an interface.

In solving neutron transport, the boundary conditions require certain considerations; whether the dimensions are finite or infinite, the surface is *reflective* or *vacuum*, and consequently, if the surface is *free* or not. The reflective characteristic of the surface is described using a quantity called *albedo*, α, which is the ratio of outgoing neutron current to incoming neutron current, i.e.

$$\begin{aligned}\alpha &= \frac{J_{out}}{J_{in}} \\ &= \frac{\displaystyle\iint_{\hat{n}\cdot\hat{\Omega}>0} d\hat{\Omega}\ \vec{\phi}(\vec{r},\hat{\Omega},E,t)}{\displaystyle\iint_{\hat{n}\cdot\hat{\Omega}<0} d\hat{\Omega}\ \vec{\phi}(\vec{r},\hat{\Omega},E,t)},\end{aligned}$$

where $\hat{n}$ is a unit vector in the direction of outward normal at position $\vec{r}$ on the surface.

A free surface is one in which no neutron enters the boundary surface, be it originating outside the boundary or *leaked* from the system. Thus, if $\vec{r}$ is a position on the boundary surface, then,

$$N(\vec{r},\hat{\Omega},E,t)=0,\ \text{if}\ \hat{n}\cdot\hat{\Omega}<0.$$

It is important to note that such conditions are satisfied by non-reentrant surfaces. Non-reentrant surface are those for which, if any two positions within the bounding surfaces are connected by a straight line, then all the points in that straight line also lies within the surfaces as well. This means that the bounding surfaces cannot be concave outwards.

2.1.4 Eigenvalues of Neutron Transport Equation

In the absence of independent sources, equation (2.31) is called the homogeneous neutron transport equation, as shown in equation (2.32).

$$\begin{aligned}\frac{1}{v}\frac{\partial}{\partial t}\phi(\vec{r},\hat{\Omega},E,t) &= -\hat{\Omega}\cdot\nabla\phi(\vec{r},\hat{\Omega},E,t) - \Sigma_t(\vec{r},E)\,\phi(\vec{r},\hat{\Omega},E,t) \\ &+ \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_t(\vec{r},E')f(\vec{r};\,\hat{\Omega}',E'\rightarrow\hat{\Omega},E)\,\phi(\vec{r},\hat{\Omega}',E',t). \quad (2.32)\end{aligned}$$

The above equation, along with boundary conditions, can be conveniently expressed as

$$\frac{\partial}{\partial t}\phi(\vec{r},\hat{\Omega},E,t) = \mathbf{L}\,\phi(\vec{r},\hat{\Omega},E,t). \quad (2.33)$$

The usual approach to solving such a equation is to assume a solution of the form

$$\phi(\vec{r},\hat{\Omega},E,t) = e^{\omega t}\,\phi(\vec{r},\hat{\Omega},E).$$

Plugging $\phi(\vec{r},\hat{\Omega},E,t)$ from the above equation in equation (2.33), we get

$$\omega\phi(\vec{r},\hat{\Omega},E) = \mathbf{L}\,\phi(\vec{r},\hat{\Omega},E,t). \quad (2.34)$$

Solving the above eigenvalue equation, it is possible to get a series solution in terms of ω_j and $\phi_j(\vec{r},\hat{\Omega},E)$ of the form

$$\phi(\vec{r},\hat{\Omega},E,t) = \sum_j e^{\omega_j t}\,\phi_j(\vec{r},\hat{\Omega},E) \quad (2.35)$$

in which ω_j are called the eigenvalues and the corresponding ϕ_j are called the eigenfunctions.

The long-term behavior of the flux will be determined by the maximum value of the *real components* of the eigenvalues. If the eigenvalue ω_0 is such that

$$\Re(\omega_0) \geq \Re(\omega_j)$$

for all j, then,

$$\phi(\vec{r},\hat{\Omega},E,t) = e^{\omega_0 t}\,\phi_0(\vec{r},\hat{\Omega},E), \text{ as } t\rightarrow\infty. \quad (2.36)$$

If ω_0 is positive, the neutron population will exponentially grow, and such a system is said to be *supercritical.* If ω_0 is negative, the neutron population will decay exponentially, and such a system is said to be *subcritical.* If ω_0 is zero, the neutron population will asymptotically approach a steady value, and the system will be called critical.

Let us note that we can rewrite equation (2.34) as

$$\hat{\Omega}\cdot\nabla\phi(\vec{r},\hat{\Omega},E,t)+\left(\Sigma_t(\vec{r},E)+\frac{\omega}{v}\right)\phi(\vec{r},\hat{\Omega},E,t)$$
$$=\oiint_{4\pi}d\hat{\Omega}'\int_0^\infty dE'\;\Sigma_t(\vec{r},E')f(\vec{r};\,\hat{\Omega}',E'\to\hat{\Omega},E)\,\phi(\vec{r},\hat{\Omega}',E',t)\quad(2.37)$$

which is satisfied by ω when $\omega=\omega_0$. Since it is possible for ω_0 to be negative, it is also possible for the second term in the left-hand side to be zero or negative, which may cause problem in numerical computations. The approach to solve such an eigenvalue problem would be to vary absorption cross section by an amount equal to $\dfrac{\omega_0}{v}$ to achieve criticality, and this affects the resulting neutron spectrum. If the eigenvalue is positive, the spectrum is *harder* than it should be and vice versa. Thus an alternative method is required that produces a more accurate spectrum useful for calculations of power distribution and neutron economy.

So, a different eigenvalue equation is required to solve the flux distribution, which can then be used to determine power distribution in a reactor more accurately. Before going into that discussion, the transport equation has to be modified. Let us recall equation (2.31), omitting independent sources.

$$\frac{1}{v}\frac{\partial}{\partial t}\phi(\vec{r},\hat{\Omega},E,t)+\hat{\Omega}\cdot\nabla\phi(\vec{r},\hat{\Omega},E,t)+\Sigma_t(\vec{r},E)\,\phi(\vec{r},\hat{\Omega},E,t)$$
$$=\oiint_{4\pi}d\hat{\Omega}'\int_0^\infty dE'\;\Sigma_t(\vec{r},E')f(\vec{r};\,\hat{\Omega}',E'\to\hat{\Omega},E)\,\phi(\vec{r},\hat{\Omega}',E',t).\quad(2.38)$$

The integral on the right-hand side can be split into two separate integrals, one catering to neutron multiplying interactions, and the other one catering to scattering interactions. In most reactor problems, the neutron multiplying interactions are fission reactions, so we can write

$$\oiint_{4\pi}d\hat{\Omega}'\int_0^\infty dE'\;\Sigma_t(\vec{r},E')f(\vec{r};\,\hat{\Omega}',E'\to\hat{\Omega},E)\,\phi(\vec{r},\hat{\Omega}',E',t)$$
$$=\oiint_{4\pi}d\hat{\Omega}'\int_0^\infty dE'\;\Sigma_f(\vec{r},E')f_f(\vec{r};\,\hat{\Omega}',E'\to\hat{\Omega},E)\,\phi(\vec{r},\hat{\Omega}',E',t)$$
$$+\oiint_{4\pi}d\hat{\Omega}'\int_0^\infty dE'\;\Sigma_s(\vec{r},E')f_s(\vec{r};\,\hat{\Omega}',E'\to\hat{\Omega},E)\,\phi(\vec{r},\hat{\Omega}',E',t).\quad(2.39)$$

Here, subscripts $_f$ and $_s$ are used to represent fission and scattering respectively. The first term on the right-hand side, representing fission source, can be modified from observing the fact that the neutrons emitted from fission can be assumed to be isotropic in the laboratory frame since it occurs after formation of a compound nucleus and the momentum of the neutron that induced the fission is negligible compared to that of the neutrons emitted. Hence the directions of outgoing neutrons, $\hat{\Omega}$ will be independent of the direction of neutron inducing the fission, $\hat{\Omega}'$. So, we can write

$$f_f(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E) = \frac{1}{4\pi}\nu(\vec{r};\, E' \to E). \tag{2.40}$$

$\nu(\vec{r};\, E' \to E)dE$ is the probability that a neutron of energy within dE about E will emerge as a result of a fission induce by a neutron of energy E'. Therefore,

$$\begin{aligned}\oiint_{4\pi} d\hat{\Omega} \int_0^\infty dE\, f_f(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E) &= \oiint_{4\pi} d\hat{\Omega} \int_0^\infty dE\;\; \frac{1}{4\pi}\nu(\vec{r};\, E' \to E)\\ &= \int_0^\infty dE\;\; \nu(\vec{r};\, E' \to E)\\ &= \bar{\nu}(r, E'). \end{aligned}\tag{2.41}$$

Thus, $\bar{\nu}(\vec{r}, E')$ is the average number of neutrons emitted by a fission induced by a neutron of energy E'. So, if we normalize the function $\nu(\vec{r};\, E' \to E)$, we get

$$\begin{aligned}\nu(\vec{r};\, E' \to E) &= \frac{\nu(\vec{r};\, E' \to E)}{\bar{\nu}(\vec{r}, E')} \times \bar{\nu}(\vec{r}, E')\\ &= \chi(\vec{r}, E)\, \bar{\nu}(\vec{r}, E').\end{aligned}\tag{2.42}$$

where $\chi(E)$ is the **fission neutron spectrum**. Using equation (2.42) in equation (2.40), we can write

$$f_f(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E) = \frac{\chi(\vec{r}, E)}{4\pi}\bar{\nu}(\vec{r}, E'). \tag{2.43}$$

Using equation (2.43), equation (2.39) can be written as

$$\begin{aligned}&\oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE'\;\; \Sigma_t(\vec{r}, E') f(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)\, \phi(\vec{r}, \hat{\Omega}', E', t)\\ &\quad = \frac{\chi(\vec{r}, E)}{4\pi} \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE'\;\; \bar{\nu}(\vec{r}, E')\Sigma_f(\vec{r}, E')\, \phi(\vec{r}, \hat{\Omega}', E', t)\\ &\quad + \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE'\;\; \Sigma_s(\vec{r}, E') f_s(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)\, \phi(\vec{r}, \hat{\Omega}', E', t).\end{aligned}\tag{2.44}$$

Using equation (2.44) in equation (2.38), we can write

$$\begin{aligned}&\frac{1}{v}\frac{\partial}{\partial t}\phi(\vec{r}, \hat{\Omega}, E, t) + \hat{\Omega}\cdot\nabla\phi(\vec{r}, \hat{\Omega}, E, t) + \Sigma_t(\vec{r}, E)\, \phi(\vec{r}, \hat{\Omega}, E, t)\\ &\quad = \frac{\chi(\vec{r}, E)}{4\pi} \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE'\;\; \bar{\nu}(\vec{r}, E')\Sigma_f(\vec{r}, E')\, \phi(\vec{r}, \hat{\Omega}', E', t)\\ &\quad + \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE'\;\; \Sigma_s(\vec{r}, E') f_s(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)\, \phi(\vec{r}, \hat{\Omega}', E', t).\end{aligned}\tag{2.45}$$

We now move our discussion in deriving the k-eigenvalue equation, used to determine criticality and flux distribution in the reactor. Let us assume a pulsed neutron source, $Q_1(\vec{r}, \hat{\Omega}, E, t)$, is used to initiate reactor operation at time $t = 0$. The neutrons from this source will be addressed as the first generation neutrons, denoted by $\phi_1(\vec{r}, \hat{\Omega}, E, t)$, which will get lost by fission, capture and leakage. Since fission is not the source for this generation, the transport equation applicable here is

$$\begin{aligned}\frac{1}{v}\frac{\partial}{\partial t}\phi_1(\vec{r}, \hat{\Omega}, E, t) + \hat{\Omega}\cdot\nabla\phi_1(\vec{r}, \hat{\Omega}, E, t) + \Sigma_t(\vec{r}, E)\,\phi_1(\vec{r}, \hat{\Omega}, E, t) \\ = \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_s(\vec{r}, E') f_s(\vec{r}; \hat{\Omega}', E' \to \hat{\Omega}, E)\,\phi_1(\vec{r}, \hat{\Omega}', E', t) \\ + Q_1(\vec{r}, \hat{\Omega}, E, t). \quad (2.46)\end{aligned}$$

Integrating the above equation from 0 to ∞ with respect to time t, the first term on the left-hand side becomes

$$\begin{aligned}\frac{1}{v}\int_0^\infty \frac{\partial}{\partial t}\phi_1(\vec{r}, \hat{\Omega}, E, t)\,dt &= \frac{1}{v}\left\{\phi_1(\vec{r}, \hat{\Omega}, E, \infty) - \phi_1(\vec{r}, \hat{\Omega}, E, 0)\right\} \\ &= 0. \quad (2.47)\end{aligned}$$

A system with only independent source and no neutron multiplication is inherently subcritical, and since the source is pulsed, the flux will slowly decay to zero as $t \to \infty$. Hence the first term in the right-hand sideof equation (2.47) is zero. The second term is also zero by the postulate that all the neutrons in first generation were introduced by the source from time $t = 0$ onward. Thus, the integral in this equation evaluates to zero.

Let us write

$$\int_0^\infty \phi_1(\vec{r}, \hat{\Omega}, E, t)\,dt \equiv \tilde{\phi}_1(\vec{r}, \hat{\Omega}, E) \quad (2.48)$$

$$\& \quad \int_0^\infty Q_1(\vec{r}, \hat{\Omega}, E, t)\,dt \equiv \tilde{Q}_1(\vec{r}, \hat{\Omega}, E). \quad (2.49)$$

Using equation (2.47), equation (2.48) and equation (2.49), the time integral of equation (2.46) becomes

$$\begin{aligned}\hat{\Omega}\cdot\nabla\tilde{\phi}_1(\vec{r}, \hat{\Omega}, E) + \Sigma_t(\vec{r}, E)\,\tilde{\phi}_1(\vec{r}, \hat{\Omega}, E) \\ = \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_s(\vec{r}, E') f_s(\vec{r}; \hat{\Omega}', E' \to \hat{\Omega}, E)\,\tilde{\phi}_1(\vec{r}, \hat{\Omega}', E'). \\ + \tilde{Q}_1(\vec{r}, \hat{\Omega}, E) \quad (2.50)\end{aligned}$$

The fissions induced by first generation neutrons results in the production of second generation neutrons. Using $\tilde{\phi}_1(\vec{r}, \hat{\Omega}, E)$, the source for the second generation of neutrons, $\tilde{Q}_1(\vec{r}, \hat{\Omega}, E)$ can be computed.

$$\tilde{Q}_2(\vec{r}, \hat{\Omega}, E) = \frac{\chi(\vec{r}, E)}{4\pi} \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \bar{\nu}(\vec{r}, E')\Sigma_f(\vec{r}, E')\, \tilde{\phi}_1(\vec{r}, \hat{\Omega}', E'). \tag{2.51}$$

Similar to equation (2.50), the transport equation for second generation neutrons can be written.

$$\begin{aligned}\hat{\Omega} \cdot \nabla\tilde{\phi}_2(\vec{r}, \hat{\Omega}, E) &+ \Sigma_t(\vec{r}, E)\, \tilde{\phi}_2(\vec{r}, \hat{\Omega}, E) \\ &= \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_s(\vec{r}, E')f_s(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)\, \tilde{\phi}_2(\vec{r}, \hat{\Omega}', E') \\ &\qquad + \tilde{Q}_2(\vec{r}, \hat{\Omega}, E).\end{aligned} \tag{2.52}$$

Substituting $\tilde{Q}_2(\vec{r}, \hat{\Omega}, E)$ from equation (2.51) in equation (2.52), we get

$$\begin{aligned}\hat{\Omega} \cdot \nabla\tilde{\phi}_2(\vec{r}, \hat{\Omega}, E) &+ \Sigma_t(\vec{r}, E)\, \tilde{\phi}_2(\vec{r}, \hat{\Omega}, E) \\ &= \frac{\chi(\vec{r}, E)}{4\pi} \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \bar{\nu}(\vec{r}, E')\Sigma_f(\vec{r}, E')\, \tilde{\phi}_1(\vec{r}, \hat{\Omega}', E') \\ &+ \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_s(\vec{r}, E')f_s(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)\, \tilde{\phi}_2(\vec{r}, \hat{\Omega}', E').\end{aligned} \tag{2.53}$$

Thus for ith generation neutrons, the transport equation becomes

$$\begin{aligned}\hat{\Omega} \cdot \nabla\tilde{\phi}_i(\vec{r}, \hat{\Omega}, E) &+ \Sigma_t(\vec{r}, E)\, \tilde{\phi}_i(\vec{r}, \hat{\Omega}, E) \\ &= \frac{\chi(\vec{r}, E)}{4\pi} \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \bar{\nu}(\vec{r}, E')\Sigma_f(\vec{r}, E')\, \tilde{\phi}_{i-1}(\vec{r}, \hat{\Omega}', E') \\ &+ \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_s(\vec{r}, E')f_s(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)\, \tilde{\phi}_i(\vec{r}, \hat{\Omega}', E').\end{aligned} \tag{2.54}$$

Let us define a *multiplication factor* k_i, as the ratio of flux (or neutron population) of ith generation to that of $(i-1)$th generation, i.e.

$$k_i = \frac{\tilde{\phi}_i(\vec{r}, \hat{\Omega}, E)}{\tilde{\phi}_{i-1}(\vec{r}, \hat{\Omega}, E)}. \tag{2.55}$$

It is expected

$$\lim_{i\to\infty} k_i = \lim_{i\to\infty} \left[\frac{\tilde{\phi}_i(\vec{r}, \hat{\Omega}, E)}{\tilde{\phi}_{i-1}(\vec{r}, \hat{\Omega}, E)}\right] = k_{eff} \tag{2.56}$$

where k_{eff} is a constant value called the **effective multiplication factor**. Hence we can write

$$\tilde{\phi}_{i-1}(\vec{r}, \hat{\Omega}, E) = \frac{\tilde{\phi}_i(\vec{r}, \hat{\Omega}, E)}{k_{eff}}, \;\; for \;\; i \to \infty. \tag{2.57}$$

Using equation (2.57) and letting $\tilde{\phi}_i(\vec{r}, \hat{\Omega}, E) = \phi(\vec{r}, \hat{\Omega}, E)$ for $i \to \infty$, equation (2.54) can be written as

$$\begin{aligned}\hat{\Omega} \cdot \nabla\phi(\vec{r}, \hat{\Omega}, E) + \Sigma_t(\vec{r}, E)\,\phi(\vec{r}, \hat{\Omega}, E) \\ = \frac{\chi(\vec{r}, E)}{4\pi k_{eff}} \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \bar{\nu}(\vec{r}, E')\Sigma_f(\vec{r}, E')\,\phi(\vec{r}, \hat{\Omega}', E') \\ + \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_s(\vec{r}, E') f_s(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)\,\phi(\vec{r}, \hat{\Omega}', E'). \quad (2.58)\end{aligned}$$

Equation (2.58) is known as the k-eigenvalue equation of neutron transport. Here k_{eff} is called an auxiliary eigenvalue. Taking the scattering term to the left-hand side, we can write

$$\mathbb{T}\mathbf{\Phi} = \frac{1}{k_{eff}}\mathbb{F}\mathbf{\Phi} \quad (2.59)$$

where $\mathbb{T}$ is an operator that represents streaming, absorption and scattering, $\mathbb{F}$ is an operator that represents fission and $\mathbf{\Phi}$ represents the eigenfunction of the angular flux.

The neutron spectrum obtained by solving this eigenvalue equation provides minimal inaccuracies in calculating power distribution and breeding ratios, since it simulates varying neutrons produced per fission, $\bar{\nu}$, until criticality is achieved, rather than varying cross-sections as discussed beforehand.

2.2 Method of Characteristics

2.2.1 An Overview of The Technique

Method of Characteristics is a widely-used technique for solving linear first-order partial differential equations (PDEs) and has found applications in a variety of engineering problem-solving. In the field of nuclear engineering, it is applied to compute and predict neutronics, hydraulic transients, system stability etc as well as simulate piping system response, LOCA, containment behavior, pool swell dynamics and so on [30].

In this technique, the PDE is transformed into a group of ODEs, each with a different set of initial conditions, that can be solved numerically and in some cases, analytically. These solutions, when combined, provides solution of the PDE.

To explain the technique in a simplistic manner, the following PDE is considered.

$$A\left(x,y,z\right)\frac{\partial z}{\partial x}+B\left(x,y,z\right)\frac{\partial z}{\partial y}=C\left(x,y,z\right) \tag{2.60}$$

where $z=f(x,y)$ is a surface in 3D. The normal to $f(x,y)$, $\vec{n}$ is

$$\begin{aligned}\vec{n}&=\langle f_x(x,y), f_y(x,y), -1\rangle\\&=\left\langle\frac{\partial z}{\partial x},\frac{\partial z}{\partial y},-1\right\rangle.\end{aligned} \tag{2.61}$$

Another way to present equation (2.60) is to write

$$\left\langle\frac{\partial z}{\partial x},\frac{\partial z}{\partial y},-1\right\rangle\cdot\langle A,B,C\rangle=0. \tag{2.62}$$

This means that the vector $\langle A,B,C\rangle$ is perpendicular to $\vec{n}$ and hence must be tangential to $f(x,y)$ at all points. Hence the integral curves generated by the vector field $\langle A,B,C\rangle$ in 3D space also lie on the surface and thus $f(x,y)$ can be represented as an union of these integral curves, also called *characteristic curves.* An arbitrary characteristic curve can be generated using an arbitrary initial point (x_0,y_0) within the domain of f by solving the set of differential equations:

$$\frac{dx}{d\mu}=A(x,y,z), \tag{2.63}$$

$$\frac{dy}{d\mu}=B(x,y,z). \tag{2.64}$$

Solving these equations generate solutions $x(\mu)$ and $y(\mu)$ which represent the characteristic curve for (x_0,y_0). Substituting the above equations in equation (2.60), we get

$$\begin{aligned}\frac{dz}{d\mu}&=C\left(x\left(\mu\right),y\left(\mu\right),z\right)\\&=C(\mu,z).\end{aligned} \tag{2.65}$$

Since equation (2.65) is a fist-order ODE, it can be solved very conveniently. A series of solutions can be generated using different (x_0,y_0). By selectively choosing an appropriate set of (x_0,y_0), the solutions generated can be combined to represent the surface $z=f(x,y)$.

2.2.2 Derivation of Integral Equation for Neutron Transport

Let us take note that in equation (2.31), the first two terms on the left-hand side is a linear combination of first order partial differentials. Hence, using *method of* *characteristics*, an integral equation can be derived using these two terms. Let s be the distance traveled by the packet of neutrons in the direction $\hat{\Omega}$. Then,

$$v = \frac{ds}{dt} \tag{2.66}$$

$$\& \quad \hat{\Omega} = \frac{d\vec{r}}{ds}. \tag{2.67}$$

So, we can write

$$\begin{aligned} \frac{1}{v}\frac{\partial}{\partial t}\phi(\vec{r},\hat{\Omega},E,t) + \hat{\Omega}\cdot\nabla\phi(\vec{r},\hat{\Omega},E,t) &= \frac{dt}{ds}\frac{\partial}{\partial t}\phi(\vec{r},\hat{\Omega},E,t) + \frac{d\vec{r}}{ds}\cdot\nabla\phi(\vec{r},\hat{\Omega},E,t) \\ &= \frac{d}{ds}\phi(\vec{r},\hat{\Omega},E,t). \end{aligned} \tag{2.68}$$

Solving equation (2.66), we get

$$\begin{aligned} \int_{t_0}^{t} v\,dt' &= \int_0^s ds', \\ \therefore \quad t &= t_0 + \frac{s}{v}. \end{aligned} \tag{2.69}$$

Solving equation (2.67), we get

$$\begin{aligned} \int_{\vec{r}_0}^{\vec{r}} d\vec{r}' &= \int_0^s \hat{\Omega}\,ds', \\ \therefore \quad \vec{r} &= \vec{r}_0 + s\hat{\Omega}. \end{aligned} \tag{2.70}$$

Using equations (2.68)-(2.70) in equation (2.31), we can write

$$\begin{aligned} &\frac{d}{ds}\phi\left(\vec{r}_0+s\hat{\Omega},\hat{\Omega},E,t_0+\frac{s}{v}\right) + \Sigma_t(\vec{r}_0+s\hat{\Omega},E)\,\phi\left(\vec{r}_0+s\hat{\Omega},\hat{\Omega},E,t_0+\frac{s}{v}\right) \\ &\qquad = \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE'\;\; \Sigma_t(\vec{r}_0+s\hat{\Omega},E') f(\vec{r}_0+s\hat{\Omega};\,\hat{\Omega}',E'\to\hat{\Omega},E) \\ &\qquad\quad \times\phi\left(\vec{r}_0+s\hat{\Omega},\hat{\Omega}',E',t_0+\frac{s}{v}\right) + Q_{ind}\left(\vec{r}_0+s\hat{\Omega},\hat{\Omega},E,t_0+\frac{s}{v}\right). \end{aligned} \tag{2.71}$$

If we let

$$\begin{aligned} &Q\left(\vec{r}_0+s\hat{\Omega},\hat{\Omega},E,t_0+\frac{s}{v}\right) = \\ &\qquad \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE'\;\; \Sigma_t(\vec{r}_0+s\hat{\Omega},E') f(\vec{r}_0+s\hat{\Omega};\,\hat{\Omega}',E'\to\hat{\Omega},E) \\ &\qquad\quad \times\phi\left(\vec{r}_0+s\hat{\Omega},\hat{\Omega}',E',t_0+\frac{s}{v}\right) + Q_{ind}\left(\vec{r}_0+s\hat{\Omega},\hat{\Omega},E,t_0+\frac{s}{v}\right), \end{aligned} \tag{2.72}$$

then we can write equation (2.71) as

$$\begin{aligned} &\frac{d}{ds}\phi\left(\vec{r}_0+s\hat{\Omega},\hat{\Omega},E,t_0+\frac{s}{v}\right) + \Sigma_t(\vec{r}_0+s\hat{\Omega},E)\,\phi\left(\vec{r}_0+s\hat{\Omega},\hat{\Omega},E,t_0+\frac{s}{v}\right) \\ &\qquad = Q\left(\vec{r}_0+s\hat{\Omega},\hat{\Omega},E,t_0+\frac{s}{v}\right). \end{aligned} \tag{2.73}$$

This is a first order differential equation which can be easily solved by multiplying it with the integrating factor

$$\exp\left[\int_0^{s'} \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right]$$

where s' is any arbitrary point between 0 and s, from which we get

$$\begin{aligned}\exp\left[\int_0^{s'} \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right] \frac{d}{ds'}\phi\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s'}{v}\right) \\ + \Sigma_t(\vec{r}_0 + s'\hat{\Omega}, E)\,\phi\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s}{v}\right) \exp\left[\int_0^{s'} \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right] \\ = Q\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s'}{v}\right) \exp\left[\int_0^{s'} \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right]. \quad (2.74)\end{aligned}$$

The left-hand side in the above equation can be collected as a differential on the product of the flux ϕ and the integrating factor, i.e.

$$\begin{aligned}\frac{d}{ds'}\left[\exp\left[\int_0^{s'} \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right] \phi\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s'}{v}\right)\right] \\ = Q\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s'}{v}\right) \exp\left[\int_0^{s'} \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right]. \quad (2.75)\end{aligned}$$

Integrating the above equation from with respect to ds' from 0 to s, we get

$$\begin{aligned}\phi\left(\vec{r}_0 + s\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s}{v}\right) \exp\left[\int_0^{s} \Sigma_t(\vec{r}_0 + s\hat{\Omega}, E)\, ds''\right] - \phi\left(\vec{r}_0, \hat{\Omega}, E, t_0\right) \\ = \int_0^s ds'\; Q\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s'}{v}\right) \exp\left[\int_0^{s'} \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right] \\ = \int_0^s ds'\; Q\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s'}{v}\right) \exp\left[-\int_{s'}^{0} \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right]. \\ (2.76)\end{aligned}$$

Rearranging the above equation, we get

$$\begin{aligned}\phi\left(\vec{r}_0 + s\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s}{v}\right) = \phi\left(\vec{r}_0, \hat{\Omega}, E, t_0\right) \exp\left[-\int_0^{s} \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right] \\ + \int_0^s ds'\; Q\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s'}{v}\right) \exp\left[-\int_{s'}^{s} \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right], \quad (2.77)\end{aligned}$$

where

$$\begin{aligned}Q\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s'}{v}\right) = \\ \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE'\;\; \Sigma_t(\vec{r}_0 + s'\hat{\Omega}, E') f(\vec{r}_0 + s'\hat{\Omega};\, \hat{\Omega}', E' \to \hat{\Omega}, E) \\ \times\, \phi\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}', E', t_0 + \frac{s'}{v}\right) + Q_{ind}\left(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E, t_0 + \frac{s'}{v}\right). \quad (2.78)\end{aligned}$$

Equations (2.77) and (2.78) is the *characteristics form* of the neutron transport equation, also referred to as integral equation for neutron transport. These equations are used in *collision probability method* and *method of characteristics* for solving neutronics problems. In a manner similar to subsection 2.1.4 at page 19, the k-eigenvalue equation in characteristic form can be derived. The equations are

$$\begin{aligned}\phi(\vec{r}_0 + s\hat{\Omega}, \hat{\Omega}, E) &= \phi(\vec{r}_0, \hat{\Omega}, E) \exp\left[-\int_0^s \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right] \\ &\quad + \int_0^s ds' \;\; Q(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E)\; \exp\left[-\int_{s'}^s \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E)\, ds''\right], \qquad (2.79)\end{aligned}$$

where

$$\begin{aligned}Q(\vec{r}, \hat{\Omega}, E) &= \frac{\chi(\vec{r}, E)}{4\pi k_{eff}} \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \bar{\nu}(\vec{r}, E')\Sigma_f(\vec{r}, E')\, \phi(\vec{r}, \hat{\Omega}', E') \\ &\quad + \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_s(\vec{r}, E') f_s(\vec{r};\, \hat{\Omega}', E' \to \hat{\Omega}, E)\, \phi(\vec{r}, \hat{\Omega}', E'). \qquad (2.80)\end{aligned}$$

In the next chapter, the approximations and adaptations to these equations required for implementing method of characteristics in one dimensional and in two dimensional geometries will be discussed.

There is a simplistic way to understand the integral formulation of neutron transport. Suppose, there is an angular flux of ϕ_0 along any arbitrary direction and energy in a medium with uniform total cross-section of Σ_t. After traveling a distance of s, the angular flux will get attenuate exponentially and will become $\phi_0\, e^{-\Sigma_t s}$ as shown in Figure 2.1.

Suppose there is an angular source Q of the same energy and direction as ϕ_0 at s' distance, extending over an infinitesimally small length ds'. This will contribute to the angular flux by about

$$\frac{Q\, dV'}{dA} = Q\, ds'$$

at that location, and will attenuate exponentially as well. Thus it will contribute a flux of $Q\, ds'\, e^{-\Sigma_t(s-s')}$ at distance s from ϕ_0. Here, dV' is infinitesimally small

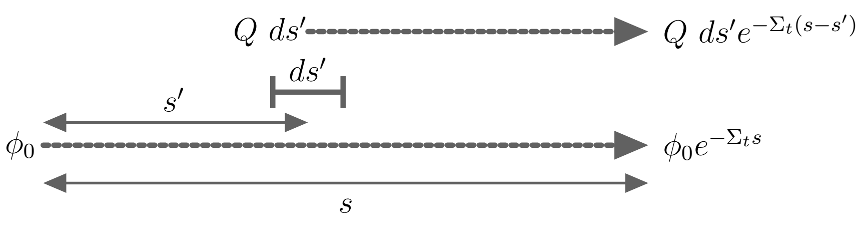


Figure 2.1: Integral Formulation of Neutron Transport

volume at s' and dA is the cross-sectional area perpendicular to ds'.

If sources like Q are present all over domain $(0, s)$, we will have to add the individual contributions through out the domain, i.e. integrate the sources from 0 to s. Thus,

$$\phi = \phi_0 \, e^{-\Sigma_t s} + \int_0^s Q \, ds' \, e^{-\Sigma_t (s-s')}. \tag{2.81}$$

If the material properties are not uniform, then the attenuation factor becomes

$$\exp\left[-\int \Sigma_t(s'') \, ds''\right].$$

So, equation (2.81) becomes

$$\phi = \phi_0 \, \exp\left[-\int_0^s \Sigma_t(s'') \, ds''\right] + \int_0^s Q \, ds' \, \exp\left[-\int_{s'}^s \Sigma_t(s'') \, ds''\right]. \tag{2.82}$$

Finally, if we include angular, energy and positional dependency in the above equation, we get

$$\begin{aligned}\phi(\vec{r}_0 + s\hat{\Omega}, \hat{\Omega}, E) &= \phi(\vec{r}_0, \hat{\Omega}, E) \exp\left[-\int_0^s \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E) \, ds''\right] \\ &+ \int_0^s ds' \; Q(\vec{r}_0 + s'\hat{\Omega}, \hat{\Omega}, E) \; \exp\left[-\int_{s'}^s \Sigma_t(\vec{r}_0 + s''\hat{\Omega}, E) \, ds''\right] \end{aligned} \tag{2.83}$$

which is the same as equation (2.79).

2.3 Scientific Computation using Python

Currently, the most popular programming language is Python [31], and its application is breaching the scientific community far and wide. For beginners, it is an easy to learn language with very simple syntaxes and allows rapid development of readable algorithms. It consists of very powerful libraries [32], the most popular of which are:

- numpy: It stands for number python which permits very efficient multi-dimensional array operations with simple syntaxes.
- matplotlib: It is a powerful data visualization tool for generating 2D as well as 3D ”publication-ready” plots.

- scipy: It provides with loads of numerical routines for various purposes such as statistical analysis, signal processing, image processing, iterative solvers, numerical integration , ODE solvers, etc. It also allows use of sparse matrices in linear algebra routines.

- sympy: It stands for symbolic python. It is used for symbolic computing

- panda: It is used in data analysis

- mayavi: It is an advanced 3D visualization tool.

Python allows flexibility to develop applications in any style in powerful environments like IPython, Spyder, Jupyter notebooks, Visual Studio Code, etc. Unlike MATLAB, it is free and supported by a wide-spread community. It also allows interfacing with C,C++ and Fortran. It is a cross-platform language with much better portability.

For calculations that are computationally extensive, the likes of which has been undertaken in this project, Python has a disadvantage over C/C++. It has a very high execution time which comes at a cost of having a readable algorithm. However, writing only the critical computationally intensive part in C and then executing the code in Python can take advantage of both worlds. It also has relatively easy to use modules for parallelization as discussed in the next chapter. Parallelizing the code also offsets the disadvantage of slow execution speed to some extent.

Chapter 3

Methodology

3.1 Modifications of Integral Equation for Neutron Transport

Equations (2.79) and (2.80) as derived in the previous chapter, although exact, is not fit and convenient for solving numerically to find eigenvalue and flux distributions unless certain approximations and assumptions are applied to fully discretize the equation over the entire problem domain. The approximations that were applied in developing the algorithm are:

- multigroup approximation,
- isotropic scattering,
- discrete ordinates approximation,
- uniform medium and density in a discrete spatial region,
- flat source approximation in a discrete spatial region.

Each of these approximations and assumptions are discussed in detail in the succeeding subsections. The following discussions were mostly drawn from Kochunas [6] and Knott and Yamamoto [5].

3.1.1 Multigroup Approximation

In the preceding chapter, we defined the volume-averaged group fluxes and group-constants for a specific geometry in a specific energy range in equations (2.12) and (2.14). We will now use these definitions to discretize equations (2.79) and (2.80) by energy.

Let us write equation (2.80) as

$$Q(\vec{r},\hat{\Omega},E) = Q_f(\vec{r},\hat{\Omega},E) + Q_s(\vec{r},\hat{\Omega},E) \tag{3.1}$$

where

$$Q_f(\vec{r},\hat{\Omega},E) = \frac{\chi(\vec{r},E)}{4\pi k_{eff}} \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \bar{\nu}(\vec{r},E')\Sigma_f(\vec{r},E')\,\phi(\vec{r},\hat{\Omega}',E'), \tag{3.2}$$

and

$$Q_s(\vec{r},\hat{\Omega},E) = \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_s(\vec{r},E') f_s(\vec{r};\,\hat{\Omega}',E' \to \hat{\Omega},E)\,\phi(\vec{r},\hat{\Omega}',E'). \tag{3.3}$$

Let us discretize the source terms over energy ranges (E_0, E_1), $(E_1, E_2), \cdots,$ (E_{n-1}, E_n) indexed by $g = 1, 2, \cdots, G$. Equation (3.2) can then be written as

$$\begin{aligned}
Q_f(\vec{r},\hat{\Omega},E) &= \frac{\chi(\vec{r},E)}{4\pi k_{eff}} \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \bar{\nu}(\vec{r},E')\Sigma_f(\vec{r},E')\,\phi(\vec{r},\hat{\Omega}',E') \\
&= \frac{\chi(\vec{r},E)}{4\pi k_{eff}} \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \int_{E_{g'}}^{E_{g'-1}} dE' \;\; \bar{\nu}(\vec{r},E')\Sigma_f(\vec{r},E')\,\phi(\vec{r},\hat{\Omega}',E') \\
&= \frac{\chi(\vec{r},E)}{4\pi k_{eff}} \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\phi_{g'}(\vec{r},\hat{\Omega}').
\end{aligned} \tag{3.4}$$

Integrating the above equation from E_g to E_{g-1} with respect to E, we get

$$\begin{aligned}
\int_{E_g}^{E_{g-1}} dE \;\; Q_f(\vec{r},\hat{\Omega},E) &= \frac{1}{4\pi k_{eff}} \int_{E_g}^{E_{g-1}} dE \;\; \chi(\vec{r},E) \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\phi_{g'}(\vec{r},\hat{\Omega}') \\
&= \frac{\chi_g(\vec{r})}{4\pi k_{eff}} \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\phi_{g'}(\vec{r},\hat{\Omega}').
\end{aligned} \tag{3.5}$$

Letting

$$\int_{E_g}^{E_{g-1}} dE \;\; Q_f(\vec{r},\hat{\Omega},E) = Q_f^g(\vec{r},\hat{\Omega}),$$

we can write

$$Q_f^g(\vec{r},\hat{\Omega}) = \frac{\chi_g(\vec{r})}{4\pi k_{eff}} \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\phi_{g'}(\vec{r},\hat{\Omega}'). \tag{3.6}$$

The scattering source term defined in equation (3.3) is discretized by energy in a similar manner.

$$\begin{aligned}
Q_s(\vec{r},\hat{\Omega},E) &= \oiint_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \;\; \Sigma_s(\vec{r},E') f_s(\vec{r};\,\hat{\Omega}',E' \to \hat{\Omega},E)\,\phi(\vec{r},\hat{\Omega}',E') \\
&= \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \int_{E_{g'}}^{E_{g'-1}} dE' \;\; \Sigma_s(\vec{r},E') f_s(\vec{r};\,\hat{\Omega}',E' \to \hat{\Omega},E)\,\phi(\vec{r},\hat{\Omega}',E') \\
&= \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \Sigma_s^{g'}(\vec{r}) f_s(\vec{r};\,\hat{\Omega}',g' \to \hat{\Omega},E)\,\phi_{g'}(\vec{r},\hat{\Omega}').
\end{aligned} \tag{3.7}$$

Integrating the above equation from E_g to E_{g-1} with respect to E, we get

$$\begin{aligned}\int_{E_g}^{E_{g-1}} dE\;\; Q_s(\vec{r},\hat{\Omega},E) &= \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \Sigma_s^{g'}(\vec{r}) \int_{E_g}^{E_{g-1}} dE\;\; f_s(\vec{r};\,\hat{\Omega}',g' \to \hat{\Omega},E)\,\phi_{g'}(\vec{r},\hat{\Omega}') \\ &= \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \Sigma_s^{g'}(\vec{r}) f_s(\vec{r};\,\hat{\Omega}',g' \to \hat{\Omega},g)\,\phi_{g'}(\vec{r},\hat{\Omega}') \\ &= \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r}) f_s(\vec{r};\,\hat{\Omega}' \to \hat{\Omega})\,\phi_{g'}(\vec{r},\hat{\Omega}'). \qquad (3.8)\end{aligned}$$

Letting

$$\int_{E_g}^{E_{g-1}} dE\;\; Q_s(\vec{r},\hat{\Omega},E) = Q_f^g(\vec{r},\hat{\Omega}),$$

we can write

$$Q_s^g(\vec{r},\hat{\Omega}) = \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r}) f_s(\vec{r};\,\hat{\Omega}' \to \hat{\Omega})\,\phi_{g'}(\vec{r},\hat{\Omega}'). \qquad (3.9)$$

Integrating equation (3.1) from E_g to E_{g-1} with respect to E, we can write

$$Q_g(\vec{r},\hat{\Omega}) = Q_f^g(\vec{r},\hat{\Omega}) + Q_s^g(\vec{r},\hat{\Omega}). \qquad (3.10)$$

Finally, the multigroup approximation of equation (2.79) leads to

$$\begin{aligned}\phi_g(\vec{r}_0 + s\hat{\Omega},\hat{\Omega}) &= \phi_g(\vec{r}_0,\hat{\Omega}) \exp\left[-\int_0^s \Sigma_t^g(\vec{r}_0 + s''\hat{\Omega})\,ds''\right] \\ &\quad + \int_0^s ds'\;\; Q_g(\vec{r}_0 + s'\hat{\Omega},\hat{\Omega})\, \exp\left[-\int_{s'}^s \Sigma_t^g(\vec{r}_0 + s''\hat{\Omega})\,ds''\right]. \qquad (3.11)\end{aligned}$$

3.1.2 Isotropic Scattering

If we assume scattering to be isotropic, i.e. the neutrons are equally likely to be deflected in any direction, then we can write

$$f_s(\vec{r};\,\hat{\Omega}' \to \hat{\Omega}) = \frac{1}{4\pi}. \qquad (3.12)$$

Using the above substitution in equation (3.9), we get

$$Q_s^g(\vec{r},\hat{\Omega}) = \frac{1}{4\pi} \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r})\,\phi_{g'}(\vec{r},\hat{\Omega}'). \qquad (3.13)$$

This will inhibit the solvers capability in accurately solving cases which involves significant scattering anisotropicity. However, at this preliminary stage, this approximation allowed easier development of the algorithm owing to more simplistic source term calculations.

3.1.3 Discrete Ordinates Approximation

The discrete ordinates approximation assumes that the scalar flux $\Phi_g(\vec{r})$ can be approximated by a quadrature summation of angular flux $\phi_g(\vec{r},\hat{\Omega}_m)$ at discrete directions $\hat{\Omega}_m$, i.e.

$$\begin{aligned}\Phi_g(\vec{r}) &= \oiint_{4\pi} d\hat{\Omega}\ \ \phi_g(\vec{r},\hat{\Omega}) \\ &= \sum_{m=1}^{M} \iint_{S_m} d\hat{\Omega}\ \ \phi_g(\vec{r},\hat{\Omega}), \qquad (3.14)\end{aligned}$$

where S_m for $m = 1, 2, \cdots, M$ are piece-wise connected surfaces that wraps a whole sphere such that

$$\begin{aligned}\sum_{m=1}^{M} \iint_{S_m} d\hat{\Omega} &= \sum_{m=1}^{M} \omega_m \\ &= 4\pi. \qquad (3.15)\end{aligned}$$

If

$$\begin{aligned}\iint_{S_m} d\hat{\Omega}\ \ \phi_g(\vec{r},\hat{\Omega}) &= \omega_m \phi_g(\vec{r},\hat{\Omega}_m) \\ &= \omega_m \phi_{g,m}(\vec{r}) \qquad (3.16)\end{aligned}$$

for a specific direction $\hat{\Omega}_m$, we can write

$$\Phi_g(\vec{r}) = \sum_{m=1}^{M} \omega_m \phi_{g,m}(\vec{r}). \qquad (3.17)$$

Similarly, we can write

$$\begin{aligned}q_f^g(\vec{r}) &= \oiint_{4\pi} d\hat{\Omega}\ \ Q_f^g(\vec{r},\hat{\Omega}) \\ &= \sum_{m=1}^{M} \omega_m Q_f^{g,m}(\vec{r}), \qquad (3.18) \\ q_s^g(\vec{r}) &= \oiint_{4\pi} d\hat{\Omega}\ \ Q_s^g(\vec{r},\hat{\Omega}) \\ &= \sum_{m=1}^{M} \omega_m Q_s^{g,m}(\vec{r}), \qquad (3.19) \\ \& \quad q_g(\vec{r}) &= \oiint_{4\pi} d\hat{\Omega}\ \ Q_g(\vec{r},\hat{\Omega}) \\ &= q_f^g(\vec{r}) + q_s^g(\vec{r}), \qquad (3.20)\end{aligned}$$

where

$$\begin{aligned}
Q_f^{g,m}(\vec{r}) &= Q_f^g(\vec{r},\hat{\Omega}_m) \\
&= \frac{\chi_g(\vec{r})}{4\pi k_{eff}} \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\phi_{g'}(\vec{r},\hat{\Omega}') \\
&= \frac{\chi_g(\vec{r})}{4\pi k_{eff}} \sum_{g'=1}^{G}\sum_{m'=1}^{M} \nu\Sigma_f^{g'}(\vec{r})\,\omega_{m'}\phi_{g',m'}(\vec{r}) \\
&= \frac{\chi_g(\vec{r})}{4\pi k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\Phi_{g'}(\vec{r}),
\end{aligned} \tag{3.21}$$

and

$$\begin{aligned}
Q_s^{g,m}(\vec{r}) &= Q_s^g(\vec{r},\hat{\Omega}_m) \\
&= \frac{1}{4\pi} \oiint_{4\pi} d\hat{\Omega}' \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r})\,\phi_{g'}(\vec{r},\hat{\Omega}') \\
&= \frac{1}{4\pi} \sum_{g'=1}^{G}\sum_{m'=1}^{M} \Sigma_s^{g'\to g}(\vec{r})\,\omega_{m'}\phi_{g',m'}(\vec{r}) \\
&= \frac{1}{4\pi} \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r})\,\Phi_{g'}(\vec{r}).
\end{aligned} \tag{3.22}$$

Using equations (3.21) and (3.15) in equation (3.18),we get

$$\begin{aligned}
q_f^g(\vec{r}) &= \sum_{m=1}^{M} \omega_m \times \frac{\chi_g(\vec{r})}{4\pi k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\Phi_{g'}(\vec{r}) \\
&= 4\pi \times \frac{\chi_g(\vec{r})}{4\pi k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\Phi_{g'}(\vec{r}) \\
&= \frac{\chi_g(\vec{r})}{k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\Phi_{g'}(\vec{r}).
\end{aligned} \tag{3.23}$$

In a similar manner, using equations (3.22) and (3.15) in equation (3.19),we get

$$q_s^g(\vec{r}) = \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r})\,\Phi_{g'}(\vec{r}). \tag{3.24}$$

Substituting the source terms from equations (3.23) and (3.24) in equation (3.20), we get

$$q_g(\vec{r}) = \frac{\chi_g(\vec{r})}{k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\Phi_{g'}(\vec{r}) + \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r})\,\Phi_{g'}(\vec{r}). \tag{3.25}$$

For $\hat{\Omega} = \hat{\Omega}_m$, we can write equation (3.10) as

$$Q_g(\vec{r},\hat{\Omega}_m) = Q_f^g(\vec{r},\hat{\Omega}_m) + Q_s^g(\vec{r},\hat{\Omega}_m),$$

or,

$$Q_{g,m}(\vec{r}) = Q_f^{g,m}(\vec{r}) + Q_s^{g,m}(\vec{r}). \tag{3.26}$$

Using equations (3.21), (3.22) and (3.25) in the above equation, we get

$$\begin{aligned} Q_{g,m}(\vec{r}) &= \frac{\chi_g(\vec{r})}{4\pi k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\Phi_{g'}(\vec{r}) + \frac{1}{4\pi} \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r})\,\Phi_{g'}(\vec{r}) \\ &= \frac{1}{4\pi}\left\{ \frac{\chi_g(\vec{r})}{k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\vec{r})\,\Phi_{g'}(\vec{r}) + \sum_{g'=1}^{G} \Sigma_s^{g'\to g}(\vec{r})\,\Phi_{g'}(\vec{r}) \right\} \\ &= \frac{q_g(\vec{r})}{4\pi}. \end{aligned} \tag{3.27}$$

The relation established above can also be deduced intuitively by observing that all interactions leading to neutron sources have been considered isotropic after our assumption in Subsection 3.1.2 in page 30.

Finally, letting $\hat{\Omega} = \hat{\Omega}_m$, equation (3.11) becomes

$$\begin{aligned} \phi_g(\vec{r}_0 + s\hat{\Omega}_m, \hat{\Omega}_m) = \phi_g(\vec{r}_0, \hat{\Omega}_m) \exp\left[-\int_0^s \Sigma_t^g(\vec{r}_0 + s''\hat{\Omega}_m)\, ds'' \right] \\ + \int_0^s ds' \;\, Q_g(\vec{r}_0 + s'\hat{\Omega}_m, \hat{\Omega}_m)\, \exp\left[-\int_{s'}^s \Sigma_t^g(\vec{r}_0 + s''\hat{\Omega}_m)\, ds'' \right], \end{aligned}$$

or,

$$\begin{aligned} \phi_{g,m}(\vec{r}_0 + s\hat{\Omega}_m) = \phi_{g,m}(\vec{r}_0) \exp\left[-\int_0^s \Sigma_t^g(\vec{r}_0 + s''\hat{\Omega}_m)\, ds'' \right] \\ + \int_0^s ds' \;\, Q_{g,m}(\vec{r}_0 + s'\hat{\Omega}_m)\, \exp\left[-\int_{s'}^s \Sigma_t^g(\vec{r}_0 + s''\hat{\Omega}_m)\, ds'' \right]. \end{aligned} \tag{3.28}$$

3.1.4 Spatial Discretization

Spatial discretization is performed by dividing the whole geometry into small meshes in a manner so that the *medium* and *density* is uniform at all locations within any individual mesh.

Let us consider a number of neutron *"rays"* oriented towards direction $\hat{\Omega}$ traverses over mesh i. The rays are indexed by k, and the length traversed by ray k in mesh i is s. If $\vec{r}_0$ is the location from which ray k enters mesh i, then we can write

$$\phi_{g,m}(\vec{r}_0) = \phi_{g,m,i,k}^{in}, \tag{3.29}$$

$$\phi_{g,m}(\vec{r}_0 + s\hat{\Omega}_m) = \phi_{g,m,i,k}(s), \tag{3.30}$$

$$\phi_{g,m,i,k}(s_{m,i,k}) = \phi_{g,m,i,k}^{out}, \tag{3.31}$$

$$Q_{g,m}(\vec{r}_0 + s\hat{\Omega}_m) = Q_{g,m,i}(s), \tag{3.32}$$

$$\& \quad \Sigma_t^g(\vec{r}_0 + s\hat{\Omega}_m) = \Sigma_t^{g,i}. \tag{3.33}$$

Using the above substitutions in equation (3.28), we get

$$\begin{aligned}\phi_{g,m,i,k}(s) &= \phi^{in}_{g,m,i,k}\exp(-\Sigma^{g,i}_t s) + \int_0^s ds'\ \ Q_{g,m,i}(s')\ \exp\left[-\int_{s'}^s \Sigma^{g,i}_t\, ds''\right] \\ &= \phi^{in}_{g,m,i,k}\exp(-\Sigma^{g,i}_t s) + \int_0^s ds'\ \ Q_{g,m,i}(s')\ \exp\left[-\Sigma^{g,i}_t (s-s')\right].\end{aligned}$$

At this point, we apply *flat-source approximation*, which assumes that the source term is approximately equal at all points within the mesh i, i.e. $Q_{g,m,i}(s) \approx Q_{g,m,i}\ \forall\ s$. This also means that the scalar flux is assumed to be flat inside a mesh, i.e. $\Phi_{g,i}(s) \approx \Phi_{g,i}$ Using this substitution in the above equation, we get

$$\begin{aligned}\phi_{g,m,i,k}(s) &= \phi^{in}_{g,m,i,k}\exp(-\Sigma^{g,i}_t s) + Q_{g,m,i}\int_0^s ds'\ \ \exp\left[-\Sigma^{g,i}_t (s-s')\right] \\ &= \phi^{in}_{g,m,i,k}\exp(-\Sigma^{g,i}_t s) + \frac{Q_{g,m,i}}{\Sigma^{g,i}_t}\ \left[1-\exp\left(-\Sigma^{g,i}_t s\right)\right]. \qquad (3.34)\end{aligned}$$

If the ray exits mesh i at $s = s_{m,i,k}$, we can write

$$\begin{aligned}\phi^{out}_{g,m,i,k} &= \phi_{g,m,i,k}(s_{m,i,k}) \\ &= \phi^{in}_{g,m,i,k}\exp(-\Sigma^{g,i}_t s_{m,i,k}) + \frac{Q_{g,m,i}}{\Sigma^{g,i}_t}\ \left[1-\exp\left(-\Sigma^{g,i}_t s_{m,i,k}\right)\right]. \qquad (3.35)\end{aligned}$$

The average angular flux, $\phi_{g,m,i,k}$ over length $s_{m,i,k}$ is

$$\begin{aligned}\phi_{g,m,i,k} &= \frac{\displaystyle\int_0^{s_{m,i,k}} ds\ \ \phi_{g,m,i,k}(s)}{s_{m,i,k}} \\ &= \frac{\displaystyle\int_0^{s_{m,i,k}} ds\ \ \phi^{in}_{g,m,i,k}\exp(-\Sigma^{g,i}_t s) + \int_0^{s_{m,i,k}} ds\ \ \frac{Q_{g,m,i}}{\Sigma^{g,i}_t}\ \left[1-\exp\left(-\Sigma^{g,i}_t s\right)\right]}{s_{m,i,k}} \\ &= \frac{\phi^{in}_{g,m,i,k}\dfrac{1-\exp(-\Sigma^{g,i}_t s_{m,i,k})}{\Sigma^{g,i}_t} + \dfrac{Q_{g,m,i}}{\Sigma^{g,i}_t}\left[s_{m,i,k} - \left\{\dfrac{1-\exp(-\Sigma^{g,i}_t s_{m,i,k})}{\Sigma^{g,i}_t}\right\}\right]}{s_{m,i,k}} \\ &= \frac{\phi^{in}_{g,m,i,k}\left[1-\exp(-\Sigma^{g,i}_t s_{m,i,k})\right] + Q_{g,m,i}\left[s_{m,i,k} - \left\{\dfrac{1-\exp(-\Sigma^{g,i}_t s_{m,i,k})}{\Sigma^{g,i}_t}\right\}\right]}{\Sigma^{g,i}_t s_{m,i,k}} \\ &= \frac{\phi^{in}_{g,m,i,k} - \left[\phi^{in}_{g,m,i,k}\exp(-\Sigma^{g,i}_t s_{m,i,k}) + \dfrac{Q_{g,m,i}}{\Sigma^{g,i}_t}\ \left\{1-\exp\left(-\Sigma^{g,i}_t s_{m,i,k}\right)\right\}\right]}{\Sigma^{g,i}_t s_{m,i,k}} \\ &\qquad + \frac{Q_{g,m,i}}{\Sigma^{g,i}_t}. \qquad (3.36)\end{aligned}$$

Substituting $\phi^{out}_{g,m,i,k}$ from equation (3.35) in the above expression, we get

$$\phi_{g,m,i,k} = \frac{\phi^{in}_{g,m,i,k} - \phi^{out}_{g,m,i,k}}{\Sigma^{g,i}_t s_{m,i,k}} + \frac{Q_{g,m,i}}{\Sigma^{g,i}_t}. \qquad (3.37)$$

Letting

$$\phi^{in}_{g,m,i,k} - \phi^{out}_{g,m,i,k} = \Delta\phi_{g,m,i,k}, \tag{3.38}$$

we can write

$$\phi_{g,m,i,k} = \frac{\Delta\phi_{g,m,i,k}}{\Sigma^{g,i}_t s_{m,i,k}} + \frac{Q_{g,m,i}}{\Sigma^{g,i}_t}. \tag{3.39}$$

Substituting $\phi^{out}_{g,m,i,k}$ from equation (3.38) to equation (3.35), we can write

$$\phi^{in}_{g,m,i,k} - \Delta\phi_{g,m,i,k} = \phi^{in}_{g,m,i,k}\exp(-\Sigma^{g,i}_t s_{m,i,k}) + \frac{Q_{g,m,i}}{\Sigma^{g,i}_t}\left[1 - \exp\left(-\Sigma^{g,i}_t s_{m,i,k}\right)\right].$$

Subjecting $\Delta\phi_{g,m,i,k}$ in the above equation and rearranging, we get

$$\Delta\phi_{g,m,i,k} = \left(\phi^{in}_{g,m,i,k} - \frac{Q_{g,m,i}}{\Sigma^{g,i}_t}\right)\left[1 - \exp\left(-\Sigma^{g,i}_t s_{m,i,k}\right)\right]. \tag{3.40}$$

The merit of finding $\Delta\phi_{g,m,i,k}$ is that it simplifies finding outward angular flux in *ray tracing* as well as the calculation of scalar flux in a mesh, decreasing overall computational load.

If the rays are all equally spaced, then average angular flux $\phi_{g,m,i}$ can be found by averaging angular fluxes $\phi_{g,m,i,k}$ determined by equation (3.39) for all k that traverses through mesh i.

$$\begin{aligned}\phi_{g,m,i} &= \frac{\sum\limits_{k\in i}\phi_{g,m,i,k}\, s_{m,i,k}}{\sum\limits_{k\in i} s_{m,i,k}} \\ &= \frac{\sum\limits_{k\in i}\Delta\phi_{g,m,i,k}}{\Sigma^{g,i}_t\sum\limits_{k\in i} s_{m,i,k}} + \frac{Q_{g,m,i}}{\Sigma^{g,i}_t}.\end{aligned} \tag{3.41}$$

Using the above equation, letting $\Phi_g(\vec{r}) = \Phi_{g,i}$, equation (3.17) can be written as

$$\begin{aligned}\Phi_{g,i} &= \sum_{m=1}^{M}\omega_m\phi_{g,m,i} \\ &= \sum_{m=1}^{M}\omega_m\left[\frac{\sum\limits_{k\in i}\Delta\phi_{g,m,i,k}}{\Sigma^{g,i}_t\sum\limits_{k\in i} s_{m,i,k}} + \frac{Q_{g,m,i}}{\Sigma^{g,i}_t}\right] \\ &= \sum_{m=1}^{M}\frac{\sum\limits_{k\in i}\omega_m\Delta\phi_{g,m,i,k}}{\Sigma^{g,i}_t\sum\limits_{k\in i} s_{m,i,k}} + \frac{1}{\Sigma^{g,i}_t}\sum_{m=1}^{M}\omega_m Q_{g,m,i} \\ &= \frac{q_{g,i}}{\Sigma^{g,i}_t} + \sum_{m=1}^{M}\frac{\sum\limits_{k\in i}\omega_m\Delta\phi_{g,m,i,k}}{\Sigma^{g,i}_t\sum\limits_{k\in i} s_{m,i,k}}.\end{aligned} \tag{3.42}$$

If we let $q_g(\vec{r}) = q_{g,i}$, $\chi_g(\vec{r}) = \chi_{g,i}$, $\Sigma_f^{g'}(\vec{r}_0+s\hat{\Omega}_m) = \Sigma_f^{g',i}$ and $\Sigma_s^{g'\to g}(\vec{r}_0+s\hat{\Omega}_m) = \Sigma_s^{g'\to g,i}$ for s within the interval $(0, s_{m,i,k})$, equation (3.25) can be written as

$$q_{g,i} = \frac{\chi_{g,i}}{k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_f^{g',i}\,\Phi_{g',i} + \sum_{g'=1}^{G} \Sigma_s^{g'\to g,i}\,\Phi_{g',i}. \tag{3.43}$$

The angular source term becomes

$$Q_{g,m,i} = \frac{q_{g,i}}{4\pi}. \tag{3.44}$$

Equations (3.38), (3.40), (3.42), (3.43) and (3.44) form the skeleton of the MOC algorithm. The iterative scheme and the implementation of MOC algorithm in one dimension and two dimensions will be discussed in the successive sections.

3.2 Iterative Scheme

3.2.1 Computation of Flux Distribution for One-Group Constant Source

A number of neutron rays are initiated at the periphery of the geometry adequately representing all directions in a manner so that each mesh in the geometry gets traversed by at least one ray in every direction considered. The rays are initiated with zero angular flux. The scalar flux values,Φ_i are initiated with $\frac{q_i}{\Sigma_t^i}$ for all i as per equation (3.42). As a single ray crosses a mesh, the decrease in angular flux (which is negative if angular flux increase), $\Delta\phi_{m,i,k}$, is calculated using equation (3.40).

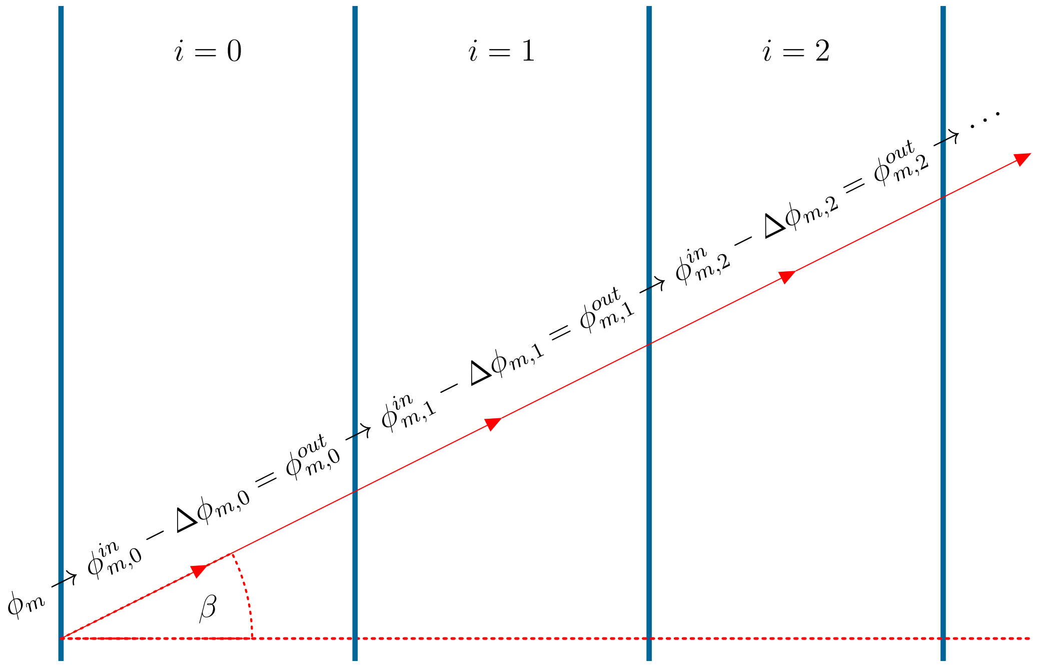
$$\Delta\phi_{m,i,k} = \left(\phi_{m,i,k}^{in} - \frac{Q_{m,i}}{\Sigma_t^i}\right)\left[1 - \exp\left(-\Sigma_t^i s_{m,i,k}\right)\right]. \tag{3.45}$$

From equation (3.42), it can be seen that the scalar flux contribution due to sweeping of ray $\phi_{m,k}$ in mesh i is $\frac{\omega_m \Delta\phi_{m,i,k}}{\Sigma_t^i \sum_{k\in i} s_{m,i,k}}$. Hence, Φ_i is incremented using the following equation:

$$\Phi_i = \Phi_i + \frac{\omega_m \Delta\phi_{m,i,k}}{\Sigma_t^i \sum_{k\in i} s_{m,i,k}}. \tag{3.46}$$

As ray k leaves mesh i, it enters mesh i' adjacent to mesh i as shown in Figure 3.1. So, we can write

$$\begin{aligned}\phi_{m.i'.k}^{in} &= \phi_{m.i.k}^{out} \\ &= \phi_{m.i.k}^{in} - \Delta\phi_{m,i,k}.\end{aligned} \tag{3.47}$$



$i = 0$
$i = 1$
$i = 2$
$\phi_m \rightarrow \phi^{in}_{m,0} - \Delta\phi_{m,0} = \phi^{out}_{m,0} \rightarrow \phi^{in}_{m,1} - \Delta\phi_{m,1} = \phi^{out}_{m,1} \rightarrow \phi^{in}_{m,2} - \Delta\phi_{m,2} = \phi^{out}_{m,2} \rightarrow \cdots$
β

Figure 3.1: Ray Sweeping

So, for convenience, we simply initiate angular flux variable $\phi_{m,k}$ and decrement it by $\Delta\phi_{m,i,k}$ for each mesh i.e.

$$\phi_{m,k} = \phi_{m,k} - \Delta\phi_{m,i,k}. \tag{3.48}$$

In a similar manner, the scalar flux Φ_i is incremented for $i = i'$ and all other successive meshes traversed by ray k. When ray k hits a boundary surface at mesh N, it is multiplied by the albedo α_N of the surface. Using this value, a ray k' will be initiated with direction $\Omega_{m'}$ which will represent the reflective counterpart of ray k at that surface. Hence we can write

$$\phi^{in}_{m'.N.k'} = \phi^{out}_{m.N.k}\,\alpha_N. \tag{3.49}$$

Once all ray k for all direction Ω_m are swept in the manner discussed above, a *transport sweep* is completed which generates a solution of Φ_i for each i. The angular fluxes at the bounding surfaces where the ray terminates, $\phi^{in}_{m.N.k}$, after a transport sweep are used to initiate rays for the next transport sweep. The scalar flux values,Φ_i are re-initiated with $\frac{q_i}{\Sigma^i_t}$ for all i in before every transport sweep. Depending on the boundary conditions, the scalar flux solution converges in a Jacobian manner after every sweep. The solution at the end of each sweep is compared with the solution before each sweep, as well as the angular flux at the boundary, until they are almost equal and the residual error is within a certain level of tolerance. The final values of Φ_i obtained after convergence is achieved is the solution of scalar flux for the fixed source problem.

3.2.2 k-Eigenvalue Problem

In this case, the solution comprises of an eigenvalue, k_{eff}, and its corresponding eigenfunction, Φ.

Let us recall equation (2.59).

$$\mathbb{T}\boldsymbol{\Phi} = \frac{1}{k_{eff}}\mathbb{F}\boldsymbol{\Phi}.$$

The following iterative scheme can be derived using the power method:

$$\boldsymbol{\Phi}^{(\mathbf{p+1})} = \mathbb{T}^{-1}\frac{1}{k_{eff}^{(p)}}\mathbb{F}\boldsymbol{\Phi}^{(p)}, \tag{3.50}$$

$$k_{eff}^{(p+1)} = k_{eff}^{(p)}\frac{\|\mathbb{F}\boldsymbol{\Phi}^{(p+1)}\|}{\|\mathbb{F}\boldsymbol{\Phi}^{(p)}\|}. \tag{3.51}$$

This means using an initial guess of the flux $\boldsymbol{\Phi}$ and k_{eff}, the fission source term is to be calculated. Now, total fission neutrons produced in p th iteration is

$$\begin{aligned} q_f^{(p)} &= \sum_i^I \int_{V_i} d^3r \sum_g^G \nu\Sigma_f^{g,i}\,\Phi_{g,i}^{(p)} \\ &= \sum_i^I \sum_g^G \nu\Sigma_f^{g,i}\,\Phi_{g,i}^{(p)}\,V_i. \end{aligned} \tag{3.52}$$

Hence we can write

$$\begin{aligned} k_{eff}^{(p+1)} &= k_{eff}^{(p)}\frac{q_f^{(p+1)}}{q_f^{(p)}} \\ &= k_{eff}^{(p)}\frac{\sum_i^I \sum_g^G \nu\Sigma_f^{g,i}\,\Phi_{g,i}^{(p+1)}\,V_i}{\sum_i^I \sum_g^G \nu\Sigma_f^{g,i}\,\Phi_{g,i}^{(p)}\,V_i}. \end{aligned} \tag{3.53}$$

In the algorithm, at first, we assume the flux is flat at all locations, and assign it to be unity, i.e. $\Phi_i^{(1)} = 1$ for all i. We also assign the initial eigenvalue to be unity i.e. $k_{eff}^{(1)} = 1$. Then we compute the source term with this flux. Let us recall equation (3.43)

$$q_{g,i} = \frac{\chi_{g,i}}{k_{eff}}\sum_{g'=1}^G \nu\Sigma_f^{g',i}\,\Phi_{g',i} + \sum_{g'=1}^G \Sigma_s^{g'\to g,i}\,\Phi_{g',i}. \tag{3.54}$$

A scalar flux solution $\Phi_{i,g}^{(2)}$ is computed by treating the source computed above, $q_{g,i}$ as a constant fixed source using the scheme discussed in the previous subsection for each group g individually. This is equivalent to finding $\mathbb{T}^{-1}\frac{1}{k_{eff}^{(1)}}\mathbb{F}\boldsymbol{\Phi}^{(1)}$. The value for $k_{eff}^{(2)}$ is calculated with this flux using equation (3.53). In a similar manner, $\Phi_{g,i}^{(3)}$, $\Phi_{g,i}^{(4)}$, $\cdots$, and $k_{eff}^{(3)}$, $k_{eff}^{(4)}$, $\cdots$, are calculated until the eigenvalue k_{eff} converges within a set tolerance.

The above iteration scheme works well specifically for cases in which the domain consists of a single energy group. This scheme is summarized in Algorithm 4 on page 112 for one-group eigenvalue problems. For problems with multiple groups, this iteration scheme leads to convergence in a Jacobian fashion. However, if the source term is updated for in-group scattering before each transport sweep, the convergence occurs in a Gauss-Seidal fashion.

This modification would require that we calculate the portion of the source term $q_{ind}^{g,i}$ independent of in-group scattering , i.e

$$q_{ind}^{g,i} = \frac{\chi_{g,i}}{k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_f^{g',i}\,\Phi_{g',i} + \sum_{g'=1,g'\neq g}^{G} \Sigma_s^{g'\to g,i}\,\Phi_{g',i} \tag{3.55}$$

instead of $q_{g,i}$ for each outer iteration. During the calculation of flux of each energy group, the source term is calculated repeatedly with the updated flux before each transport sweep.

$$q_{g,i} = q_{ind}^{g,i} + \Sigma_s^{g\to g,i}\,\Phi_{g,i}. \tag{3.56}$$

This source term is then used to perform the transport sweep. Though flux convergence will require a higher number of transport sweeps in each outer iteration due to repetitive source update, the total number of outer iterations will be reduced by large factor as will be shown in the next chapter and thus the overall run-time will decrease with in-scattering update of the source. However, such an scheme will lead to higher run-time for problems that constitute single energy group. This scheme is summarized in Algorithm 5 on page 113.

3.3 Implementation in One Dimension

During ray tracing, at least a single ray should pass through each mesh for each direction considered. So a single ray is sufficient to trace over all the meshes in one dimension for each direction and so the index k can be dropped in equations (3.45)-(3.49) in this case. Let the size of mesh i in one dimension be l_i. If the direction Ω_m is defined by a polar angle of θ_p and an azimuthal angle of β_b as

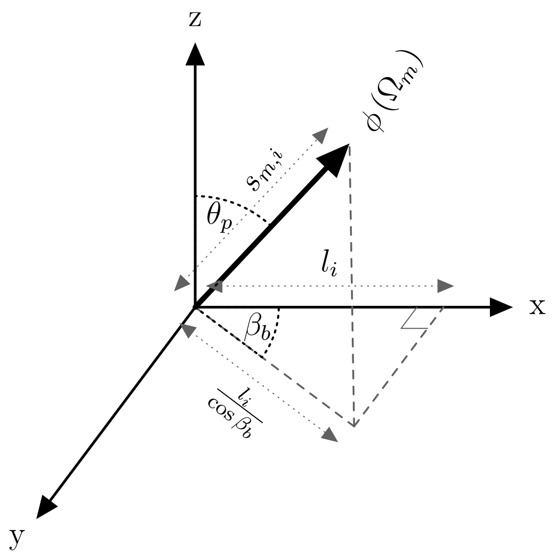


Figure 3.2: Determining length of ray from mesh size

shown in Figure 3.2, then,

$$\begin{aligned} s_{m,i} &= s_{b,p,i} \\ &= \frac{l_i}{\sin\theta_p\, \cos\beta_b}. \end{aligned} \tag{3.57}$$

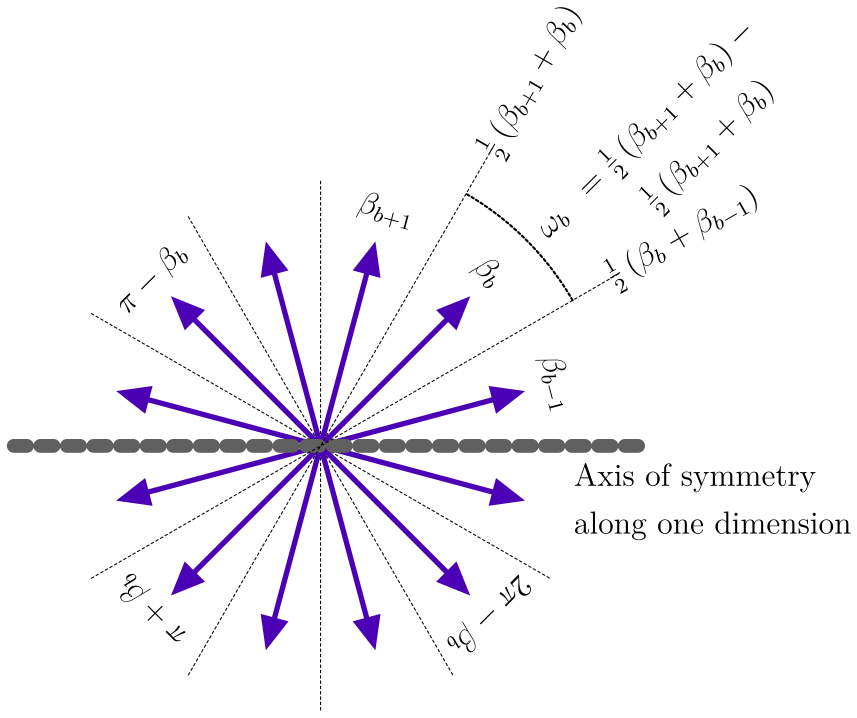
The associated weight can be resolved as

$$\begin{aligned} \omega_m &= \iint_{S_m} d\hat{\Omega} \\ &= \iint_{S_m} \sin\theta\ d\theta d\beta \\ &= \int_{\theta_p} \sin\theta\, d\theta \int_{\beta_b} d\beta \\ &= \omega_p\, \omega_b. \end{aligned} \tag{3.58}$$

Here, ω_p and ω_b are weights associated with polar angle θ_p and azimuthal angle β_b.

For implementation in one dimension as well as in two dimensions,the azimuthal division represented by β_b is assumed to be bounded by the lines bisecting the angle it forms with the adjacent directions β_{b-1} and β_{b+1}, as shown in the figure. Hence, we can write

$$\begin{aligned} \omega_b &= \frac{1}{2}(\beta_b + \beta_{b+1}) - \frac{1}{2}(\beta_{b-1} + \beta_b) \\ &= \frac{1}{2}(\beta_{b+1} - \beta_{b-1}). \end{aligned} \tag{3.59}$$



$\frac{1}{2}(\beta_{b+1}+\beta_b)$
$\omega_b = \frac{1}{2}(\beta_{b+1}+\beta_b) - \frac{1}{2}(\beta_{b+1}+\beta_b)$
$\frac{1}{2}(\beta_b+\beta_{b-1})$
β_{b+1}
β_b
β_{b-1}
$\pi - \beta_b$
$\pi + \beta_b$
$2\pi - \beta_b$
Axis of symmetry along one dimension

Figure 3.3: Azimuth Directions in Ray Tracing

The azimuthal dependency varies with geometry, and so usually, a significant number of azimuthal divisions has to be considered in the range 0 to 2π. However, a ray traced in β direction is similar to a ray traced in $2\pi - \beta$ direction, since it is the reflection of β on the axis of symmetry in one dimension (see Figure 3.3). So ray tracing is performed for β between 0 and π. If the forward tracing covers a line with azimuth direction β in the range 0 to $\frac{\pi}{2}$, then backward tracing covers azimuth direction $\pi - \beta$, reflection of $\pi + \beta$ as can be observed in the figure, in the range $\frac{\pi}{2}$ to π. Hence tracing rays with azimuth directions, β from 0 to $\frac{\pi}{2}$ in both forward and backward directions cover the whole domain from 0 to 2π.

In one dimension, it is convenient to choose β to be *uniformly distributed* i.e. they are all equally spaced, because this will mean that the weight ω_b associated with each azimuth direction β_b are equal for all b. So, if we allocate β from 0 to $\frac{\pi}{2}$ in B divisions, then the azimuth directions are

$$\beta_b = \left(b - \frac{1}{2}\right)\frac{\pi}{2B} \tag{3.60}$$

for $b = 1, 2, \cdots, B$. And so the associated weights are

$$\omega_b = \frac{\pi}{B}. \tag{3.61}$$

which accounts for both rays with azimuth direction β_b and its reflective counterpart $2\pi - \beta_b$.

Table 3.1: Tabuchi-Yamamoto Polar Quadrature Set

Number of Polar Divisions	$\sin\theta_p$	ω_p
1	0.798184	1.000000
2	0.363900	0.212854
	0.899900	0.787146
3	0.166648	0.046233
	0.537707	0.283619
	0.932954	0.670148

There are few choices available for polar quadrature set (θ_p, ω_p). Some of them are:

- uniformly distributed (UD) angle quadrature set,
- Gauss-Legendre (GL) quadrature set,
- Leonard's optimum (LO) quadrature set,
- Tabuchi-Yamamoto (TY) quadrature set, etc.

Though UD set is straight forward to implement, the results they produce are the least accurate. The GL set provides more accurate result, since they are evaluated so that it integrates the Legendre polynomials exactly, the higher the divisions, the more accurate the result. But still it requires a significant number of polar divisions, increasing the computational load of performing a transport sweep. The LO set and the TY set produces very accurate result with only 3 polar divisions, requiring minimal computational load. These sets are generated as an approximation to Bickley function used in collision probability method. The TY set provides even more accurate result and can be as accurate as a GL set with 16 polar divisions. The quadrature set is given in Table 3.1.

Similar to azimuth directions, tracing a ray with polar direction θ_p is identical to tracing a ray with polar direction $\pi - \theta_p$, since they are reflection of one-another on the axis of symmetry. So polar direction within 0 and $\frac{\pi}{2}$ are traced with each of the polar weights ω_p doubled in the MOC algorithm.

To perform the transport sweep, for each β_b and θ_p, a single ray $\phi_{g,m} = \phi_{g,b,p}$ is initiated. As the ray is swept over each mesh i, $s_{b,p,i}$ is calculated using equation

(3.57). $\Delta\phi_{g,b,p,i}$ is calculated :

$$\Delta\phi_{g,b,p,i} = \left(\phi_{g,b,p} - \frac{Q_{g,b,p,i}}{\Sigma_t^{g,i}}\right)\left[1 - \exp\left(-\Sigma_t^{g,i} s_{b,p,i}\right)\right]. \tag{3.62}$$

$\Phi_{g,i}$ is then incremented using the following equation:

$$\Phi_{g,i} = \Phi_{g,i} + \frac{\omega_b \omega_p \Delta\phi_{g,b,p,i}}{\Sigma_t^{g,i} s_{b,p,i}}. \tag{3.63}$$

$\phi_{g,b,p}$ is then decremented using the following equation:

$$\phi_{g,b,p} = \phi_{g,b,p} - \Delta\phi_{g,b,p,i}. \tag{3.64}$$

The ray is swept forward using equations (3.62)-(3.64) for all polar directions θ_p by incrementing i and when the ray intersects the boundary, the ray is swept backward in a similar manner by decrementing i. Once the ray returns to the starting point, the ray is terminated. At both boundary interfaces, the angular flux $\phi_{g,b,p}$ is stored in $\boldsymbol{\phi}_{g,boun.}$ to check angular flux convergence and then boundary conditions are applied on it.

The transport sweeps are performed to generate flux solutions repetitively until the qth sweep for which

$$\max\left[\frac{\boldsymbol{\Phi}_g^{(q)} - \boldsymbol{\Phi}_g^{(q-1)}}{\boldsymbol{\Phi}_g^{(q)}}\right] < 10^{-8} \tag{3.65}$$

and

$$\max\left[\frac{\boldsymbol{\phi}_{g,boun.}^{(q)} - \boldsymbol{\phi}_{g,boun.}^{(q-1)}}{\boldsymbol{\phi}_{g,boun.}^{(q)}}\right] < 10^{-8}. \tag{3.66}$$

The inner loop for transport sweeps is then broken after qth sweep and $\Phi_{g,i}^{(q)}$ is then regarded as the scalar flux solution for group g. The flux for all energy groups are solved in a similar manner and these solutions are then used to calculate k_{eff}. The overall routine is summarized in Algorithm 2 on page 110. The code for one-group solver in one dimension is also provided in Appendix B.2 on page 115, as a sample for the reader to assess how the algorithms were executed in Pyhton.

3.4 Implementation in Two Dimensions

Shifting from one dimension to two dimension, the scope of the challenge changes significantly in terms of complexity as well as the computational load. For simplicity and convenience, the code was developed only for solving rectangular geometries using rectangular meshes.

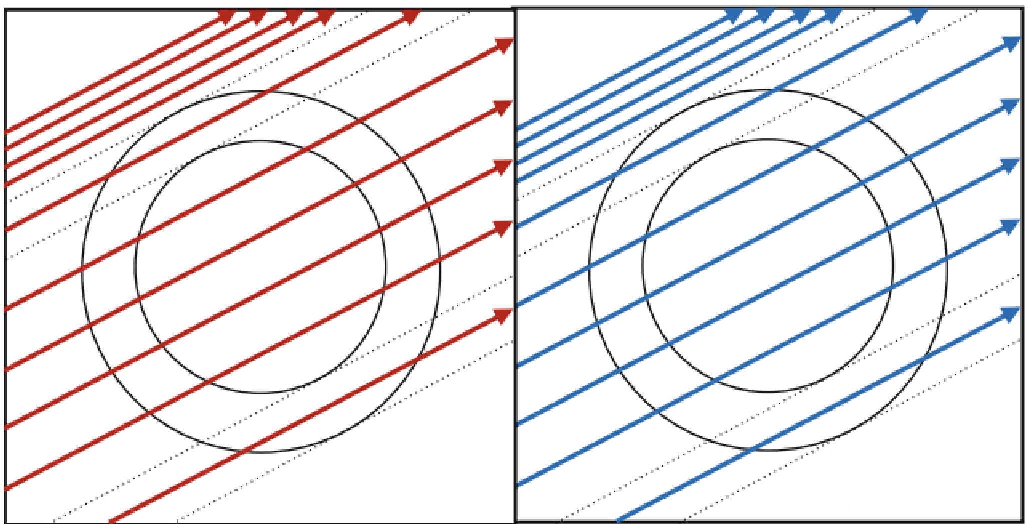


Figure 3.4: Track Layout in Macro-Band Approach [5]

Unlike our implementation in one dimension, multiple rays has to be traced for each Ω_m so that each mesh has at least one ray traversing over it in that direction. Thus the tracks along which the rays are to be traced has to be laid out at first to enable computation in an optimized manner. There are two conventional approach in laying tracks for ray tracing:

- The Macro-Band Approach,
- The Cyclic-Tracking Approach.

In Macro-Band approach, the incoming rays does not exactly align with the outgoing rays at an interface or a boundary as illustrated in Figure 3.4, so the outgoing angular flux at that interface $\phi^{out}_{g,m,k}$ has to be interpolated from the incoming rays $\phi^{in}_{g,m',k'}$. The advantage it provides is that it reduces the number of rays that needs to be traced by omitting the rays that are redundant, since it is adequate to have only one ray traversing over each mesh. The disadvantage is that it adds numerical inaccuracies from interpolation of outgoing angular flux $\phi^{out}_{g,m,k}$.

In contrast with Macro-Band approach, Cyclic-Tracking approach does not require the outgoing angular fluxes to be interpolated since the outgoing rays and incoming rays coincide exactly at the boundary or the interface. The outgoing rays at a boundary can be treated as specular reflections i.e if a ray with azimuth direction β terminates at a certain point in the boundary, then its reflective counterpart with azimuth direction $\pi - \beta$ or $2\pi - \beta$ originates exactly from that point, as can be seen in Figure 3.5. However, this would also require even ray spacing which might lead to multiple rays with same orientation traversing over a single

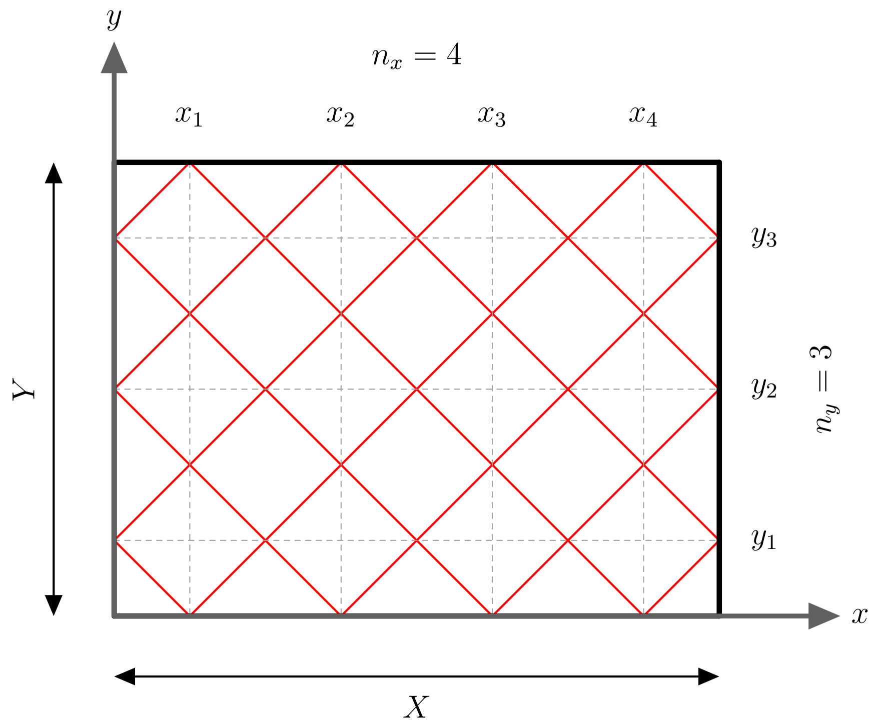


Figure 3.5: Arrangement of Tracks for an Arbitrary Direction

mesh. For the purpose of this project, cyclic-tracking approach was used owing to the simplicity in handling boundary conditions.

Laying the tracks in this approach would limit the azimuth direction,β_b that can be considered, unlike Macro-Band approach, where any arbitrary β_b can be traced. This is because $\tan\beta_b$ must be a rational number to allow for rays of directions β_b and $\pi - \beta_b$ aligning perfectly at the boundary. The arrangement in which the rays are laid out is shown in Figure 3.6.

In cyclic-tracking, even ray-spacing leads to rays meeting the boundary at equidistant points from each other for each azimuthal direction considered. If the length and the breadth of the geometry is X and Y, and n_x and n_y are number of equidistant points in which the rays meet at horizontal and vertical boundaries on either side of the geometry for a particular azimuthal angle β_b considered, then we can write

$$\delta x = \frac{X}{n_x}, \tag{3.67}$$

$$\& \quad \delta y = \frac{Y}{n_y}. \tag{3.68}$$

The rays intersect the boundaries at points $(x_u, 0)$,(x_u, Y), $(0, y_v)$ and (X, y_v)

where

$$x_u = \left(u - \frac{1}{2}\right)\delta x \quad \text{for} \quad u = 1, 2, 3, \cdots, n_x \tag{3.69}$$

$$\& \quad y_v = \left(v - \frac{1}{2}\right)\delta y \quad \text{for} \quad v = 1, 2, 3, \cdots, n_y \tag{3.70}$$

as shown in Figure 3.5.

All the points of origin and termination are displaced by $\frac{\delta x}{2}$ or $\frac{\delta y}{2}$ from the corners in order to avoid any exception to specular reflection. From Figure 3.6, we can write

$$|\sin\beta| = \frac{d}{\delta x}, \tag{3.71}$$

$$|\cos\beta| = \frac{d}{\delta y}. \tag{3.72}$$

Substituting δx and δy from equation (3.67) and equation (3.68) in above equations, we get

$$n_x = \frac{X}{d}|\sin\beta|, \tag{3.73}$$

$$n_y = \frac{Y}{d}|\cos\beta|. \tag{3.74}$$

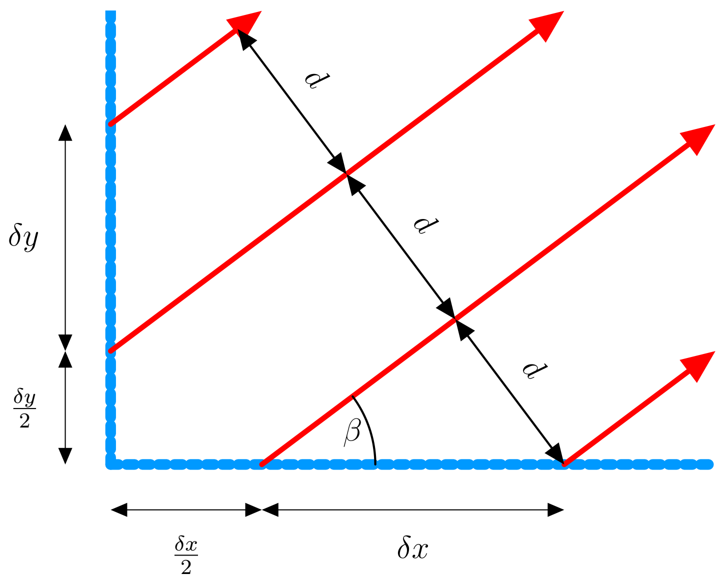


Figure 3.6: Geometric Arrangement of Track Spacing

Cyclic-tracking can not be implemented with any arbitrary value of β, since n_x and n_y must be integers, and equation (3.73) and equation (3.74) will almost always produce non-integer values. So any combination of β and d has to be slightly adjusted to allow for cyclic-tracking. For a desired combination of β^* and d^*, we first find and convert n_x and n_y by rounding it to the nearest integer.

$$n_x = \left\lfloor \frac{X}{d^*} |\sin \beta^*| \right\rceil, \tag{3.75}$$

$$n_y = \left\lfloor \frac{Y}{d^*} |\cos \beta^*| \right\rceil. \tag{3.76}$$

The values of δx and δy are now calculated from n_x and n_y using equation (3.67) and equation (3.68). The adjusted values, β and d are calculated using the following equations:

$$\beta = \arctan \frac{\delta y}{\delta x}, \tag{3.77}$$

$$d = \frac{\delta x \, \delta y}{\sqrt{\delta x^2 + \delta y^2}}. \tag{3.78}$$

Discretization of any rectangular geometry will require significant number of meshes, thus rounding of n_x and n_y will introduce a slight change in β and d to enable us consider that $\beta \approx \beta^*$ and $d \approx d^*$.

Similar to the implementation in one dimension, an uniform distribution of β_b is desirable to assure all azimuth directions are adequately considered, thus improving the accuracy. However, to enable cyclic-tracking, slight adjustments will occur as discussed just before. Hence, the azimuth quadrature set of ω_b will not exactly be equal for all β_b, but close. We first start with a desirable set of azimuth directions β_b^* of uniform distribution, i.e.

$$\beta_b^* = \left(b - \frac{1}{2}\right) \frac{\pi}{2B} \tag{3.79}$$

for $b = 1, 2, 3, \cdots, B$ where B is the number of azimuthal divisions from 0 to $\frac{\pi}{2}$.

We also choose an appropriate ray spacing d^* for all β_b^*, depending on the size of the mesh so that no mesh can fit within two adjacent ray of same orientation. Using the scheme discussed above, a set of values for β_b and d_b is generated. Using the values of β_b, the azimuthal quadrature is computed in the same manner as shown in Figure 3.3:

$$\omega_b = \frac{1}{2}(\beta_{b+1} - \beta_{b-1}) \tag{3.80}$$

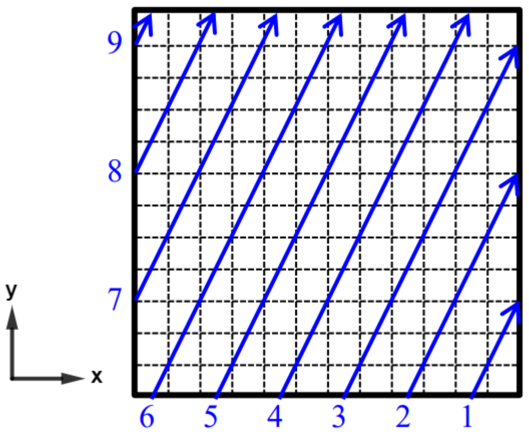


Figure 3.7: Arrangement for Sequential Sweep [6]

for $1 < b < B$. For $b = 1$,

$$\omega_1 = \frac{1}{2}\left(\beta_1 + \beta_2\right). \tag{3.81}$$

For $b = B$,

$$\omega_B = \frac{\pi}{2} - \frac{1}{2}\left(\beta_{B-1} + \beta_B\right). \tag{3.82}$$

Since the azimuth direction opposite to β_b is $\pi + \beta_b$, and after reflection, the azimuth direction of the ray changes from β_b to $\pi - \beta_b$ or $2\pi - \beta_b$, the weight for directions β_b, $\pi - \beta_b$, $\pi + \beta_b$ and $2\pi - \beta_b$ have the same value, ω_b, owing to the symmetry of ray tracing about both x-axis and y-axis, due to the inherent feature of cyclic tracking. To account for the polar distribution of the rays, we use TY quadrature set discussed in the preceding section. The polar weights ω_p are also doubled similar to the implementation in one dimension.

We now proceed our discussion to sweep algorithm. There are multiple arrangements in which rays can be swept in a two dimensional rectangular geometries. Two most common arrangements are:

- sequential ray sweeping,
- cyclic ray sweeping.

In Figure 3.7, the way in which sequential sweep proceeds is shown. For clarity, the ray spacing was made larger than the mesh size. In actual implementation, it has to be significantly smaller so that all meshes are intersected by any of the rays at least once. In the figure, we observe the projection of the rays in xy plane. All the rays drawn here are numerically labeled and will be swept following the sequence in which they are labeled. The rays labeled 1, 2 and 3 terminate at the

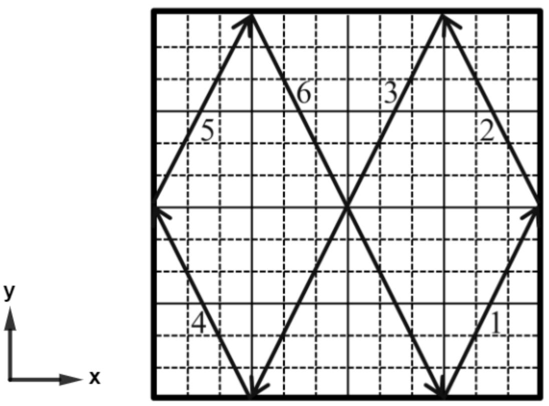


Figure 3.8: Arrangement for Cyclic Sweep [6]

vertical boundary to the right. If these rays form an angle of β_b with x-axis, the rays originating from the points of their termination will form an angle of $\pi - \beta_b$ as observed in specular reflection. So the angular flux are stored before termination after applying the boundary condition (by multiplying it with the albedo α at that surface), and these stored angular flux values are used to initiate rays with azimuth direction $\pi - \beta_b$ at the points of termination later, in the current transport sweep or the next, depending on the order of β_b in which the transport sweep proceeds. In a similar manner, all the angular fluxes at the boundary are treated for each ray. By the virtue of axial symmetry in two dimension, the polar direction remains unchanged from reflection at the bounding surfaces. Once a ray is terminated after sweeping forward, the ray is re-initiated there to be swept backward in $\pi + \beta_b$ direction, since the ray tracing data are same for both azimuth directions β_b and $\pi + \beta_b$ but in reverse order. This improves the overall cache coherency (since the ray tracing data is preferably precomputed and stored, which will be discussed later). Once all the rays with azimuth direction β_b are swept, the rays with azimuth direction β_{b+1} are initiated.

In Figure 3.8, the way in which cyclic sweep proceeds is shown. The rays are swept in the order as numbered in the figure. The point where ray 1 terminates, ray 2 originates which can be seen as an exact specular reflection of ray 1. So the angular flux at the end of ray 1 is carried by ray 2 after applying the boundary condition at the surface where the two rays are connected. So, rather than immediately terminating the ray, unlike the arrangement in sequential sweep, the ray sweeping is continued along direction 2. In a similar manner, ray 2, ray 3 and other subsequent rays are swept. By virtue of cyclic-tracking approach, after a certain number of rays are terminated, which in this figure is 6, we will return to the origin of ray 1, thus completing one whole cycle. At the end of the cycle, the ray is terminated and the angular flux is stored after applying the boundary condition , in order to initiate the cycle in the next sweep. The cycle is then swept in reverse order. It should be noted that one cycle may not be enough to traverse over all meshes in all directions (which is the case in this figure). In that case, multiple such cycles has to swept, each with different layout, consistent with the tracks laid out in cyclic approach.

Cyclic sweep has the advantage of faster convergence of angular flux in a more Gauss-Seidal fashion than sequential sweep, and also requires storing much fewer angular fluxes to initiate the next sweep. However, it has considerably lower cache coherency compared to sequential sweep since larger array of ray tracing data has to be loaded into the cache. Also sequential sweep can be performed with both macro-band approach as well as cyclic sweep approach. But cyclic sweep can only be performed with cyclic tracking approach. In this project, cyclic sweep arrangement was adopted for the simplicity in storing and calling angular flux values. The

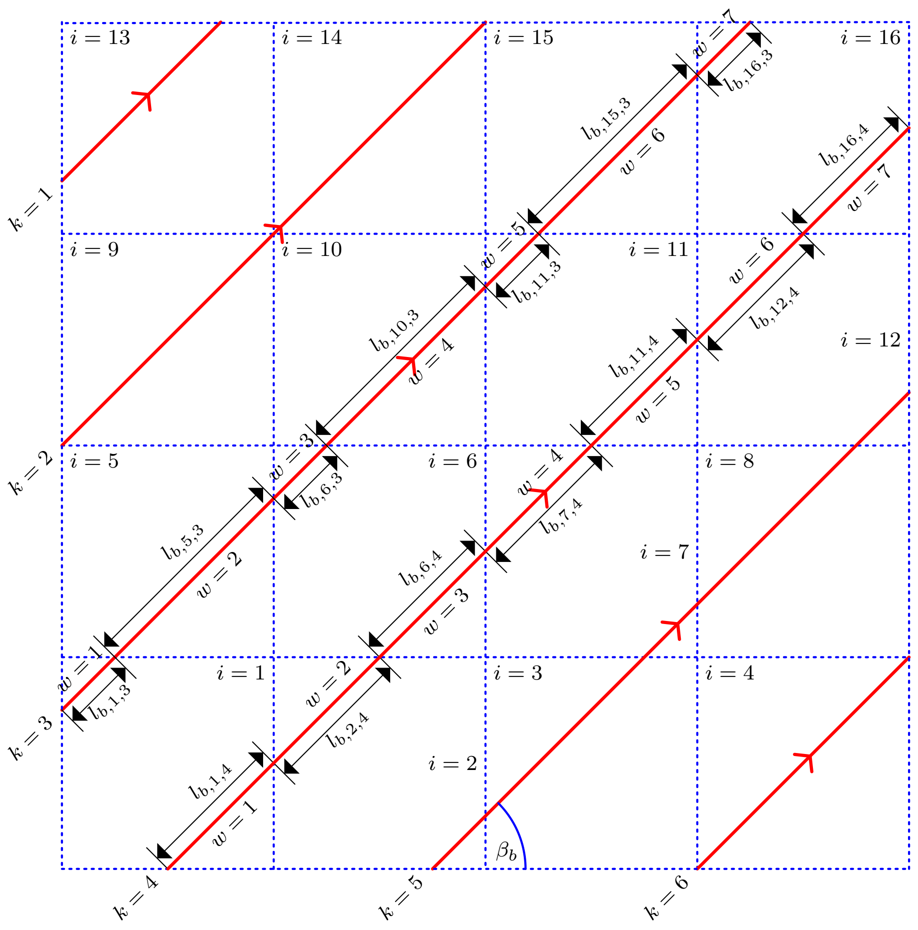


Figure 3.9: Ray Tracing in Two Dimensions

ray-tracing scheme using cyclic sweep is explained in Algorithm 1 on page 109.

As a mesh i is traversed by a ray k with orientation Ω_m defined by β_b and θ_p, the length of the ray with in the mesh is $s_{m,i,k} = s_{b,p,i,k}$ and its projection on the xy plane is $l_{b,i,k}$. The length $l_{b,i,k}$ has to be calculated first using trigonometry (and other geometric means depending on the mesh layout).

Figure 3.9 shows the manner in which ray tracing is performed in two dimensions. Here, w refers to the index of the segment of the ray k. The length of the projection on the xy plane of each ray (colored red) gets segmented by mesh partitioning lines (colored blue) as shown in the figure. Length of each of these segments, $l_{b,i,k}$ are calculated and stored under the index w in the sequence they occur. $s_{b,p,i,k}$ is calculated:

$$s_{b,p,i,k} = \frac{l_{b,i,k}}{\sin\theta_p}. \tag{3.83}$$

We then calculate $\Delta\phi_{g,b,p,i,k}$.

$$\Delta\phi_{g,b,p,i,k} = \left(\phi^{in}_{g,b,p,i,k} - \frac{Q_{g,b,p,i}}{\Sigma^{g,i}_t}\right)\left[1 - \exp\left(-\Sigma^{g,i}_t s_{b,p,i,k}\right)\right]. \tag{3.84}$$

Calculation of $\Delta\phi_{g,b,p,i,k}$ involves multiplication of two factors, one containing the source-term and angular flux, which will change repetitively after every transport sweep or source update, and the other containing exponential attenuation. Since the factor containing exponential attenuation will not change with every transport sweep or source update, it opens a window for optimization. The computational load of calculating lengths $l_{b,i,k}$ and $s_{b,p,i,k}$ and the exponential attenuation factors repetitively on the fly during transport sweep can be averted by calculating these values beforehand. This will lead to a significant reduction in run-time, but will require more memory storage, since the exponential factors has to be stored for all groups and all combinations of θ_p and β_b for each ray in each mesh they traverse through. So we first calculate the projected lengths on xy plane, $l_{b,i,k}$ for each ray using a ray tracing algorithm. All the lengths for each ray (or cycle) in successive meshes are traced and stored in an array in the same order they are intersected by the ray. The exponential attenuation factors, $E_{g,b,p,i,k}$ for each energy group is then calculated and stored in the same order as in ray tracing data.

$$\begin{aligned}E_{g,b,p,i,k} &= 1 - \exp\left(-\Sigma_t^{g,i} s_{b,p,i,k}\right) \\ &= 1 - \exp\left(-\Sigma_t^{g,i} \frac{l_{b,i,k}}{\sin\theta_p}\right).\end{aligned} \tag{3.85}$$

The iterative scheme is then initiated. For each energy group, transport sweeps are performed until a converged flux solution is obtained for the group. In performing the transport sweep, for each combination of β_b and θ_p, a ray k with angular flux $\phi_{g,b,p,k}$ is initiated for all k from the lower horizontal boundary. The ray is then swept over each mesh i it intersects using the precomputed factors $E_{g,b,p,i,k}$. First $\Delta\phi_{g,b,p,i,k}$ is calculated.

$$\Delta\phi_{g,b,p,i,k} = E_{g,b,p,i,k}\left(\phi_{g,b,p,k} - \frac{Q_{g,b,p,i}}{\Sigma_t^{g,i}}\right). \tag{3.86}$$

Noting that

$$\sum_{k\in i} s_{b,p,i,k} = \frac{1}{\sin\theta_p}\sum_{k\in i} l_{b,i,k},$$

$\Phi_{g,i}$ is thus incremented using the following equation:

$$\Phi_{g,i} = \Phi_{g,i} + \frac{\omega_b\omega_p \sin\theta_p\, \Delta\phi_{g,b,p,i,k}}{\Sigma_t^{g,i} \sum\limits_{k\in i} l_{b,i,k}} \tag{3.87}$$

which would require calculating and storing $\sum_k l_{b,i,k}$ beforehand in the ray tracing algorithm. Finally, angular flux is decremented by $\Delta\phi_{g,b,p,i,k}$.

$$\phi_{g,b,p,k} = \phi_{g,b,p,k} - \Delta\phi_{g,b,p,i,k}. \tag{3.88}$$

The ray k is swept forward using equations (3.86)-(3.88) for all polar divisions p by incrementing v, the ray index. If the ray hits the boundary, the angular flux $\phi_{g,b,p,k}$ is multiplied by the albedo α at the bounding surface. The incoming flux on each boundary interface is stored beforehand as well to check the convergence of angular flux. Once the ray completes the cycle and returns to its point of origin, the angular flux is stored to initiate the cycle in the next sweep. A second ray k' is then swept along the same cycle but backwards (in opposite direction) by decrementing v in a similar manner. Once the cycle is swept both forward and backward, the next cycle is swept until all the cycles are swept. This will result in sweeping completed for all rays with all polar directions and and azimuth directions β_b, $\pi - \beta_b$, $\pi + \beta_b$ and $2\pi - \beta_b$. So the index b is then incremented. Once sweeping is completed for all b, a transport sweep is completed, which generates a solution for the scalar flux $\Phi_{g,i}$ for all i.

The transport sweeps are performed to generate flux solutions repetitively until the qth sweep for which

$$\max\left[\frac{\mathbf{\Phi}_g^{(q)} - \mathbf{\Phi}_g^{(q-1)}}{\mathbf{\Phi}_g^{(q)}}\right] < 10^{-8} \tag{3.89}$$

and

$$\max\left[\frac{\boldsymbol{\phi}_{g,boun.}^{(q)} - \boldsymbol{\phi}_{g,boun.}^{(q-1)}}{\boldsymbol{\phi}_{g,boun.}^{(q)}}\right] < 10^{-8}. \tag{3.90}$$

The inner loop for transport sweep is then broken after qth sweep and $\mathbf{\Phi}_g^{(q)}$ is then regarded as the scalar flux solution for group g. The flux for all energy groups are solved in a similar manner and these solutions are then used to calculate k_{eff}. The overall routine is summarized in Algorithm 3 on page 111.

3.5 Parallelization Scheme for Transport Sweep Computation

Almost all of the computational load in the entire execution of the code goes into performing transport sweep. Due to the constraint of Global Interpreter Lock (GIL) in Python, the sweep operation is executed in serial and can not utilize more than one core, unless the code is parallelized. This would mean other cores present in a multi-core CPU will remain idle during the execution which, if utilized, will reduce the run-time of the code by a significant factor. Thus a problem that will take days to find the solution via serial execution can be solved in a matter of few hours if is executed in parallel. Thus parallelization of the code enables the user to solve a wider range of problems relatively faster, making it lesser time consuming than established Monte-Carlo codes.

The scope of parallelization applies to two types of situation:

- Input-Output bound
- CPU bound

Transport sweep is CPU bound, and so a process-based parallelism has to be applied. There are a number of Python modules available or applicable for parallelism. Some of them are:

- multiprocessing
- Dask

- Joblib
- IPyParallel
- mpi4py
- parallel python
- Celery

The modules listed above are applicable in different contexts [33]. In our project, we set the target to fully utilize the CPU of a typical stand-alone desktop, which usually has between two to eight cores. Hence the **multiprocessing** module was used, as the author found it more adaptable to the problem with vast resources available online [34].

The multiprocessing module allows parallelization by spawning a number of *child processes*, or a *pool of child processes*, with each child process sharing a portion of the total work-load. Pool allows one to equitably distribute the work-load into a number of processes but within certain constraints. Thus, spawning a process allows flexibility in tailoring the handling of the work-load which pool does not provide.

Conventionally, after the parent process spawns the child processes, it sleeps and waits until they are terminated (via `.join()`)and then it executes the rest of the code. Following this convention in the context of MOC algorithm means spawning the child processes repetitively for each transport sweep. Spawning either a process or a pool takes significant time. Thus, it was seen through trial that using the conventional parallelization scheme does not achieve its goal. The parallel execution code as a result of this scheme will require much longer time to execute than its serial execution counterpart.

Hence, the author had to figure out an unconventional scheme. The child processes can not be spawned in every transport sweep. The author chose to spawn these child processes only once for the entire execution of the code. This decision comes with certain limitations as follows.

- The child process can not be manually put to sleep with out a set time duration. This is because the programming tool appropriate for this hack does not exist yet (to the best of author's knowledge).
- The parent process has to execute the rest of the MOC algorithm, so it can not be put to sleep either during transport sweep with out set duration.

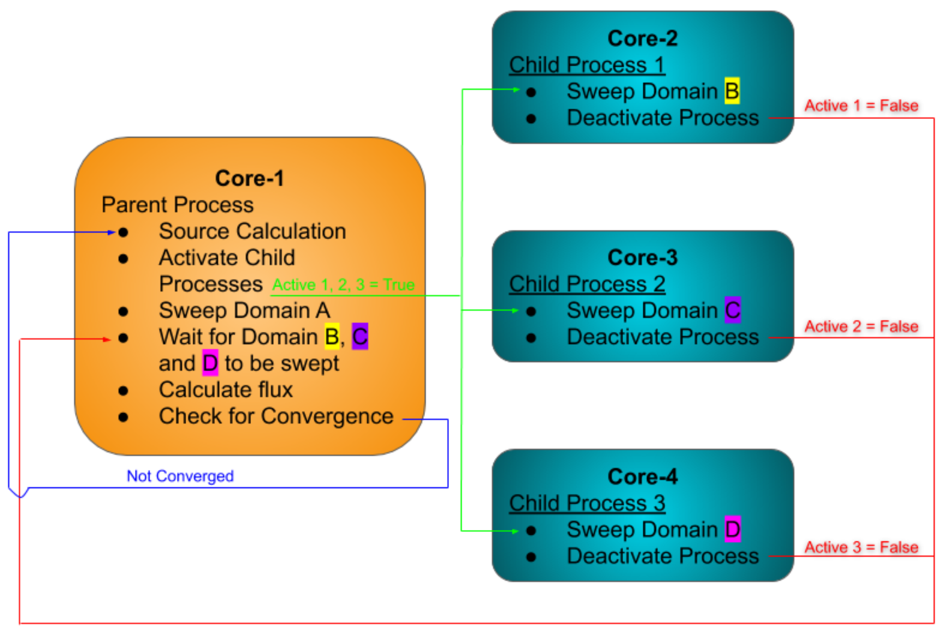


Figure 3.10: Parallelization Scheme for Transport Sweep

Thus, to achieve optimum utilization:

- one of the cores will have to be committed to the parent process, and rest of the cores to the child processes, and
- work-load of transport sweep has to be (almost) equally distributed between all processes, including parent process.

This means for a CPU with a quad-core processor (4 cores), there will be a Parent Process and three child processes: Child Process 1, Child Process 2 and Child Process 3. The work-load will be divided into 4 equal domain: Domain A, Domain B, Domain C and Domain D. Parent Process will execute over Domain A and Child Process 1, 2 and 3 will execute over Domain B, C and D respectively during a transport sweep.

The child processes would require two switches:

- a kill switch shared by all the child processes to terminate them after source convergence is achieved, and
- an individual switch for each child process to activate it to perform its task and then deactivate it. This neither puts the child process to sleep, nor kills it. It is simply a tool for synchronization.

The disadvantage of such an scheme is that the CPU will clock at its full potential redundantly when the code executes code blocks other than transport sweep. Also the sweep algorithm has to be written separately for the parent process than the child processes which does not incorporate any of the switches the child processes have. So the code block for transport sweep was written twice in the Python script.

In both 1D and 2D implementation, the domain was decomposed based on the number of azimuthal divisions B between 0 and $\frac{\pi}{2}$. This means that if B=8, then Domain A will consist of all rays with $\beta \in \{\beta_1, \beta_2\}$, Domain B will consist of all rays with $\beta \in \{\beta_3, \beta_4\}$, and so on. In the case of 2D, a more *scalable* domain decomposition is possible if rays are swept sequentially rather than in cycle, where the domain is decomposed over the number of rays, k. However, this would require more synchronization between the processes resulting in a reduction in *speedup* and is redundant in a typical desktop pc. On the other hand, decomposition based on azimuthal divisions will not be as scalable in a high configuration pc with 16 or more cores. Since the goal was to develop the code to be used in desktop pc, the author preferred to decompose over azimuthal divisions.

The decomposed variables were appended in a list. First the section of the whole domain partitioned as individual sub-domains: Domain A, Domain B, etc. were identified. The following snippet shows the method.

```python
#azim : array containing the values of azimuthal angles
#cores : number of CPU cores the code will utilize
angle_per_core=len(azim)/cores
range_list=[]  #list for appending index values
for i in range(cores+1):
    a_b=round(i*angle_per_core)
    range_list.append(a_b)
```

The domain is now divided and appended in a list. The snippet below shows an example for decomposition of azimuthal weights, ω_b.

```python
#omega_azim : vector containing the values of azimuthal weights
#omega_azim_list : list to contain the decomposed azimuthal weights
omega_azim_list=[]
for core in range(cores):
    omega_azim_list.append(omega_azim[range_list[core]:range_list[core+1]])
```

In this manner, all the required data were decomposed, namely $E_{g,b,p,i,k}$, β_b, ω_b, etc.

The multiprocessing module offers three types of tools for communication between processes: **sharedctypes variables**, **queues** and **pipes**. ”queues” are suitable for communication between multiple processes in a ”first in first out (FIFO)” basis, whereas ”pipes” are suitable for two way communication. However they are both unsuitable for handling large chunks of data, freezing the processes, and so all forms of communications were restricted to sharedctypes variables and arrays.

Both the kill switch and the activating switches would require communication from the main process, and so a shared boolean variable was used. The angular source term will also be updated repetitively, so a shared array was used. However, this was a technical issue, since multiprocessing module offers only 1D array for sharing, which complicates calling and storing values in contexts that require an array of higher dimensions.

However, this issue can be resolved by making a synchronized copy of the shared array as a numpy array . First we declare the shared array with the require number of elements.

```
#Ny or Nx : number of meshes along y or x direction
import multiprocessing as mp
Q_mp=mp.RawArray('d',Ny*Nx) #Q_mp : shared array to pass angular source term
                                #data
```

We then create a synchronized copy of this array as a numpy array with desired dimensions in all the processes required.

```
import numpy as np
Q=np.asarray(Q_mp).reshape((Ny,Nx)) #Q : numpy array to read and write data
```

Thus changing an element in numpy array ”Q” of parent process will automatically write the change in shared array ”Q_ mp” and thus in numpy array ”Q” of the child processes. In a similar manner, data of scalar flux and angular flux at the boundary are sent from child processes to the parent process. However, for sharing scalar flux data, each child process has its own shared array. This is because if they use the same shared array, the child processes may overwrite the same element simultaneously, leading to error. To avoid this race condition, further synchronization is necessary, which was averted by allocating separate shared array for each child process.

To evaluate the performance of a parallelized code, two metrics are determined: *Speedup* and $Efficiency$. Another relevant metric is *Scaling*, used to compare a parallelized code's adaptability in High Performance Computing.

Speedup is the ratio of time it takes to solve a problem in serial execution with a single core, T_s to the time it takes to solve the same problem in parallel execution with κ cores, T_κ.

$$Speedup = \frac{T_s}{T_\kappa} \tag{3.91}$$

$Efficiency$ is the ratio of *Speedup* to number of cores κ used to obtain that *Speedup*.

$$Efficiency = \frac{Speedup}{\kappa} \tag{3.92}$$

In the next chapter, the performance and fidelity of the algorithms used in 1D and 2D implementation as discussed in this chapter would be probed and analyzed by solving different benchmark problems and comparing the results along with sensitivity analysis to test the soundness of the code.

Chapter 4

Results and Discussion

4.1 One-Group Eigenvalue Solver in One Dimension

At first, the solver was benchmarked using homogeneous slabs. The problem set was taken from an article by Modak [35], which defines two slabs of same material but of different length, one with a length of unit mfp (mean free path) and the other with a length of 8 mfp, with vacuum boundary conditions on both sides. The cross-section data are given in Table 4.1. Sensitivity analysis of the solver was performed by varying the number of meshes and the azimuthal divisions. The convergence pattern was observed with and without in-scattering update during transport sweep. The results are presented in Table 4.2

Next, the solver was benchmarked using heterogenous slabs. The problem set was taken from an article by Kornreich [36]. Each slab consists of seven regions with each region having a width of an unit mfp. Case 1 consists of fuel and reflector and Case 2 and 3 consists of fuel, reflector and absorber. The configurations are shown in Figure 4.1 and the cross section data are also given in Table 4.1. Sensitivity analysis was performed by varying only mesh numbers with 64 azimuthal divisions. The convergence pattern was observed with and without scattering update during transport sweep. The results are presented in Table 4.3.

For homogeneous slab of 1 mfp width, the error in the eigenvalue decreases as number of meshes increases and azimuthal divisions increases. However, the error is heavily dependent on azimuthal divisions. But with enough mesh divisions and azimuthal divisions, the residual error due to discretization can be brought down to zero as seen in the table. It can be speculated that with a mesh size of 0.1 mfp and 16 azimuthal divisions between 0 and 2π, the residual error is very small

Table 4.1: Cross Section Data for One-Group Problem Sets in One Dimension

Problem Set		Σ_t (cm^{-1})	Σ_s (cm^{-1})	$\nu\Sigma_f$ (cm^{-1})
Homogenous Slab		1.0	0.8	1.0
Seven Region Heterogeneous Slab:	Fuel	0.415	0.334	0.178
	Reflector	0.371	0.334	0.0
	Absorber	0.371	0.037	0.0

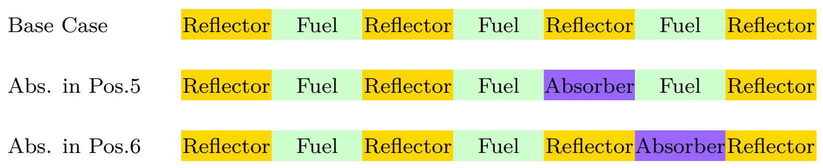

Base Case
Reflector Fuel Reflector Fuel Reflector Fuel Reflector
Abs. in Pos.5
Reflector Fuel Reflector Fuel Absorber Fuel Reflector
Abs. in Pos.6
Reflector Fuel Reflector Fuel Reflector Absorber Reflector

Figure 4.1: Seven Region Heterogeneous Slab Arrangements

Table 4.2: Results for One-Group Benchmark Problems in One Dimension: Homogeneous Slabs

Case	Azimuthal Divisions	Mesh Divisions	k_{eff}	Error (%)
1 mfp width			1.2264 (reference value)	
	4	1	0.9805	20.1
	4	10	1.0007	18.4
	4	100	1.0020	18.3
	16	100	1.2175	0.73
	64	100	1.2260	0.03
	256	100	1.2263	0.01
	1024	1000	1.2264	0.00
8 mfp width			4.2300 (reference value)	
	16	1	3.7436	11.5
	16	8	4.1229	2.53
	16	80	4.2280	0.05
	16	800	4.2295	0.01
	64	800	4.2300	0.00

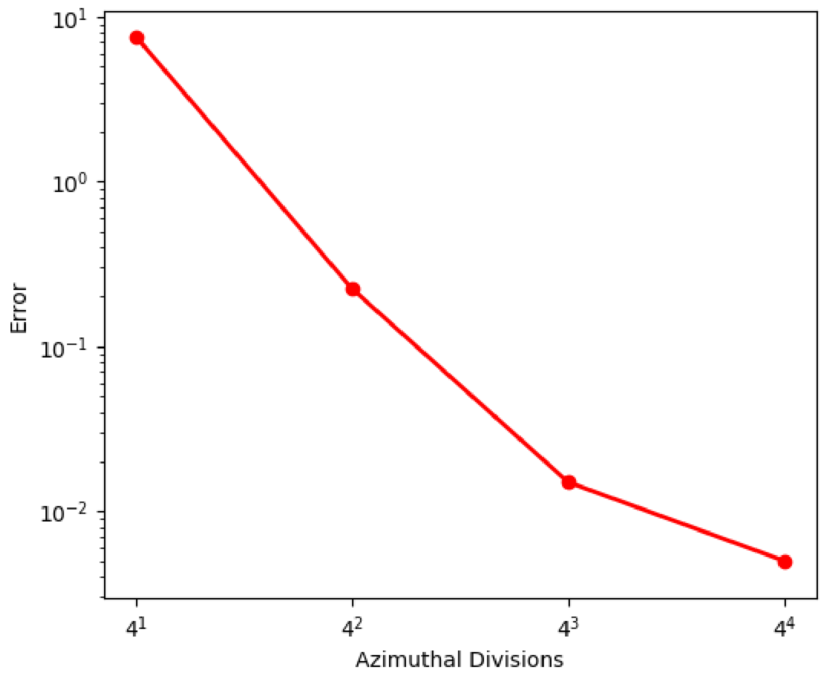
Table 4.3: Results for One-Group Benchmark Problems
in One Dimension : Seven Region Heterogeneous Slab

Case	Azimuthal Divisions	Mesh Divisions in Each Region	k_{eff}	Error (%)
Base case			1.17361 (reference value)	
	4	1	1.08493	7.556
	4	5	1.12813	3.875
	4	25	1.13063	3.662
	4	125	1.13073	3.653
	16	125	1.17237	0.106
	64	125	1.17352	0.008
	256	125	1.17357	0.003
Abs. in Pos.5			0.94268 (reference value)	
	4	125	0.87191	7.507
	16	125	0.94058	0.223
	64	125	0.94254	0.015
	256	125	0.94263	0.005
Abs. in Pos.6			1.02265 (reference value)	
	256	1	0.97116	5.035
	256	5	1.01864	0.392
	256	25	1.02244	0.021
	256	125	1.02261	0.004

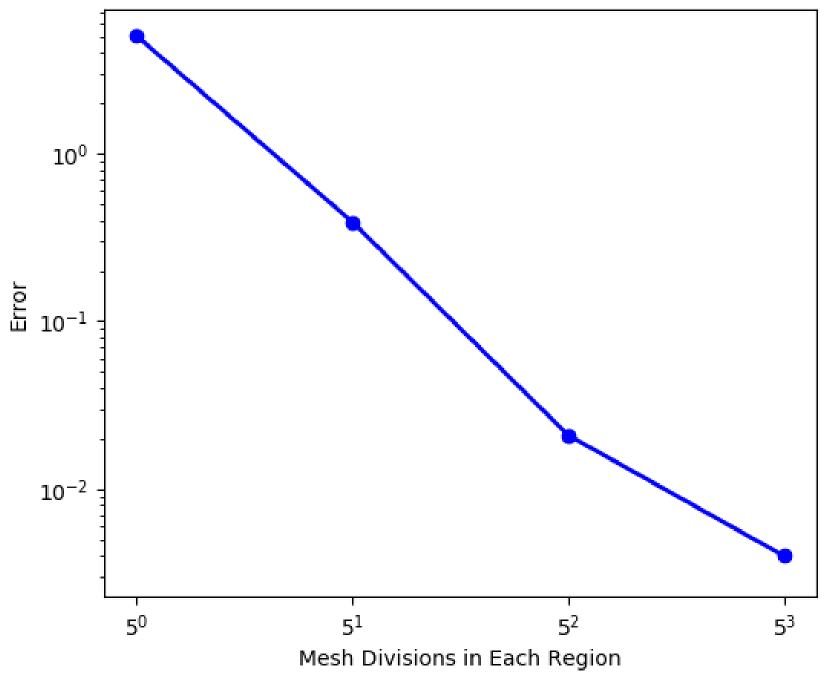
within 1%.

In the case of 8 mfp width, we observe that the error due to mesh size can be significant (over 10% with a single mesh). This is due to flat source approximation, which gets minimized significantly when mesh size is 1 mfp or less. However, the residual error can be minimized to zero with much lower azimuthal divisions, compared to 1 mfp width case. The smaller geometry allows more leakage, which might have caused the stronger azimuthal dependency of the 1 mfp width case.

In the seven region benchmark, significant error can be observed almost equally



(a) Graph of Error vs Azimuthal Divisions for Seven Region Heterogeneous Geometry: Abs. in Pos.5 Case



(b) Graph of Error vs Number of Mesh Divisions for Seven Region Heterogeneous Geometry: Abs. in Pos.6 Case

Figure 4.2: Sensitivity of Error to Azimuthal Divisions and Mesh Divisions

Table 4.4: Effect of Source Update for Scattering After Each Transport Sweep on Convergence

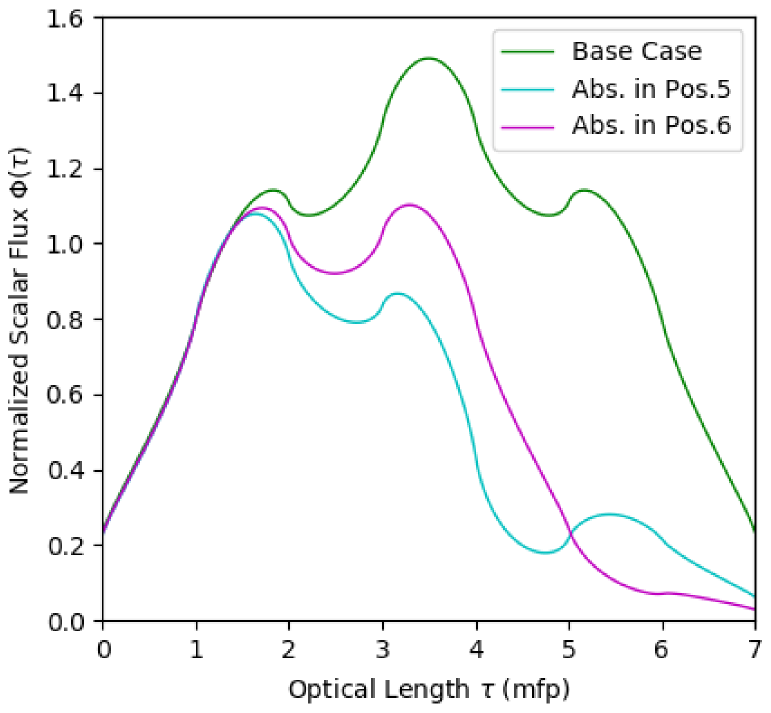
Case	Without Scattering Source Update		With Scattering Source Update	
	Transport Sweeps	Source Iterations	Transport Sweeps	Source Iterations
Base Case	156	78	685	21
Abs. in Pos.5	254	127	1381	42
Abs. in Pos.6	136	68	453	14

dependent on both azimuthal divisions and mesh divisions in each regions. Likewise, by increasing the number of mesh divisions and azimuthal divisions, the residual error can be brought down to be negligible. Figure 4.2a and Figure 4.2b show how the error strictly depends on the number of azimuthal divisions and mesh divisions. The graphs are almost linear, suggesting a power relationship between the error and number of azimuthal divisions or mesh divisions.

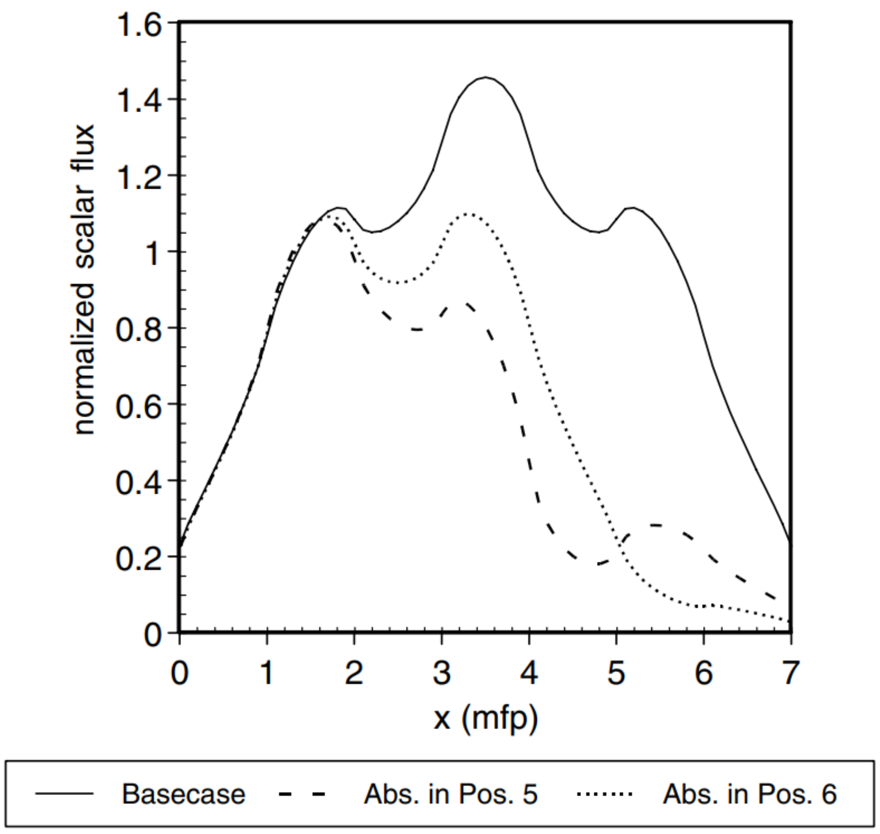
In Table 4.4, the effect of updating scattering source after each transport sweep can be observed. All the data were generated with 64 azimuthal divisions and 125 mesh divisions in each region. The total number of source iterations required for convergence decreases by a significant factor. However, the number of transport sweeps required per source iteration increases by about 16 times in each case. Hence the overall number of transport sweeps and thus computational load increases. Thus, such an iterative scheme is undesirable for one-group solver. However, as will be shown in the next section, this iterative scheme results in faster convergence in multigroup solver when updated for in-scattering only.

Finally, we compare the flux distribution for each of the three cases with the reference distribution. The flux distributions generated by the solver with 64 azimuthal divisions and 125 mesh divisions in each region are shown in Figure 4.3a. The reference distributions are shown in Figure 4.3b. As can be observed from the graphs, the distributions are identical.

By implementing the method of characteristics in one dimension for one-group problems, it was observed how the error depends on mesh size and azimuthal divisions. The proper iterative scheme for one-group problems was also justified which will be implemented in two dimension one-group solver. The flux distribution pro-



(a) Flux Distributions by Python MOC Solver



(b) Reference Flux Distributions from Kornreich and Parsons [36]

Figure 4.3: Flux Distributions for Three Cases of Seven Region Heterogeneous Slab Benchmark Problem

duced by the solver were also consistent. These exercises helped in understanding and establishing the principles in applying this method. In the next section, further exercises done for multigroup benchmark problems in one dimension will be discussed before moving on to two dimensional problems to gain more fluency and confidence in the method.

4.2 Multigroup Eigenvalue Solver in One Dimension

Four benchmark problems taken from an article by Sood [37] were used to benchmark the solver, of which, two are homogeneous infinite medium and two are heterogeneous slab lattice cell. The first benchmark problem, URR-3-0-IN consisted of three energy groups in a homogeneous infinite medium. It served to evaluate the capability of the solver to accurately solve a multigroup system irrespective of the geometry. The cross section data are provided below in Table 4.5. All cross section data are provided in cm^{-1}.

The second benchmark, URR-6-0-IN was an extension to the first benchmark but consisted of six energy groups, in which the first three groups are decoupled from the last three groups, except for χ which evenly distributes the neutrons on both sides. Thus the resulting solution will be the same as the first benchmark. The six group benchmark served to evaluate the solver's performance to incorporate up-scattering. The cross section data are provided below in Table 4.6. All cross section data are provided in cm^{-1}.

Both these benchmarks were solved for a slab of 2 cm with reflective boundary conditions using 100 mesh divisions and 64 azimuthal divisions. The results of three group and six group infinite medium benchmarks are reported in Table 4.8 and in Table 4.9. As seen in the tables, the exact solutions of the eigenvalues and the flux ratios were generated by the solver, which attests to the solver's reliability

Table 4.5: Cross Section Data for Three Group Infinite Medium Benchmark

g	χ_g	ν_g	Σ_f^g	Σ_t^g	$\Sigma_s^{1\to g}$	$\Sigma_s^{2\to g}$	$\Sigma_s^{3\to g}$
1	0.96	3.0	0.006	0.24	0.024	0.0	0.0
2	0.04	2.5	0.06	0.975	0.171	0.6	0.0
3	0.0	2.0	0.9	3.1	0.033	0.275	2.0

Table 4.6: Cross Section Data for Six Group Infinite Medium Benchmark

g	χ_g	ν_g	Σ_f^g	Σ_t^g	$\Sigma_s^{1\to g}$	$\Sigma_s^{2\to g}$	$\Sigma_s^{3\to g}$	$\Sigma_s^{4\to g}$	$\Sigma_s^{5\to g}$	$\Sigma_s^{6\to g}$
1	0.48	3.0	0.006	0.240	0.024	0.0	0.0	0.0	0.0	0.0
2	0.02	2.5	0.06	0.975	0.171	0.6	0.0	0.0	0.0	0.0
3	0.0	2.0	0.9	3.10	0.033	0.275	2.0	0.0	0.0	0.0
4	0.0	2.0	0.9	3.10	0.0	0.0	0.0	2.0	0.275	0.033
5	0.02	2.5	0.06	0.975	0.0	0.0	0.0	0.0	0.60	0.171
6	0.48	3.0	0.006	0.240	0.0	0.0	0.0	0.0	0.0	0.024

Table 4.7: Cross Section Data for Heterogeneous Slab Lattice Cell Benchmark

Medium	g	χ_g	ν_g	Σ_f^g	Σ_t^g	$\Sigma_s^{1\to g}$	$\Sigma_s^{2\to g}$
Research	1	1	1.004	0.61475	0.650917	0	0
Reactor (d)	2	0	2.5	0.045704	2.138	0.0342008	2.0688
H_2O (b)	1	0	0	0	0.110683291	0.109674215	0
	2	0	0	0	0.36355	0.001000596	0.36339
H_2O (c)	1	0	0	0	1.331518007	1.226381244	0
	2	0	0	0	4.3735	0.104639534	4.3547

in handling multigroup problems, with and without up-scattering.

The next two benchmarks consist of two mediums: fuel and moderator. The cross section data are provided in Table 4.7. The third benchmark, URRd-H_2Ob(1)-2-0-ISLC consists of Research Reactor (d) of length 0.0329074 cm and H_2O (b) of length 9.034787 cm. The fourth benchmark,URRd-H_2Oc(1)-2-0-ISLC consists of Research Reactor (d) of length 0.341011 cm and H_2O (c) of length 0.751023 cm. Both benchmarks are exactly at critical condition ($k_\infty = 1$). The benchmarks were solved using 64 azimuthal divisions and 100 mesh divisions in each region. The result from the solver has a negligible error for the third benchmark and solves the fourth benchmark exactly. The results are shown in Table

Table 4.8: Result of Three Group Infinite Medium Benchmark

	k_∞	$\frac{\Phi_2}{\Phi_1}$	$\frac{\Phi_3}{\Phi_1}$	$\frac{\Phi_3}{\Phi_2}$
Solver	1.600000	0.480000	0.15	0.312500
Reference	1.600000	0.480	0.150	0.3125

Table 4.9: Result of Six Group Infinite Medium Benchmark

	k_∞	$\frac{\Phi_2}{\Phi_1}$	$\frac{\Phi_3}{\Phi_1}$	$\frac{\Phi_3}{\Phi_2}$	$\frac{\Phi_5}{\Phi_6}$	$\frac{\Phi_4}{\Phi_6}$	$\frac{\Phi_4}{\Phi_5}$
Solver	1.600000	0.480000	0.15	0.312500	0.480000	0.15	0.312500
Reference	1.600000	0.480	0.150	0.3125	0.480	0.15	0.3125

Table 4.10: Results for Heterogeneous
Slab Lattice Cell Benchmark

Benchmark	k_∞	Error (%)
URRd-H_2Ob(1)-2-0-ISLC	0.99998	2.04×10^{-3}
URRd-H_2Oc(1)-2-0-ISLC	1.00000233	2.33×10^{-4}

4.10. This attests to the solver's capability to handle multigroup problems in heterogeneous geometry as well.

The lattice cell benchmarks were designed to stress the solvers capability, and thus required a large number of transport sweeps to achieve convergence. This made these benchmarks ideal case to test the iterative schemes for multigroup problems as well as the solvers efficiency in parallelization. We first tested the effect on convergence for updating the source for inscattering only after every transport sweep. The criteria for group flux convergence was set to 10^{-12} and for the source convergence was set to 10^{-10}. The results are reported in Table 4.11.

As can be seen in the table, the number of source iterations falls by a large factor as a result of in-scattering update. The decrease in total source iterations was also observed in one-group problems. However, unlike one-group problems, there is a significant reduction in total number of transport sweeps performed. This is because the source from in-scattering is independent of other group fluxes and thus can be updated independently, leading to convergence in a Gauss-Seidal fashion. On the other hand, in one-group problems, the total source, comprising fission and scattering, is totally dependent only on single group flux solution, so updating the whole source through the source iteration rather than scattering only leads to faster convergence.

We then tested the efficacy of the parallelization scheme in one dimension. The computations were performed using Intel(R) Core(TM) i5-4460 CPU @3.20GHz

Table 4.11: Effect of Source Update for In-scattering After Each Transport Sweep on Convergence

Case	Without In-scattering Source Update		With In-scattering Source Update	
	Transport Sweeps	Source Iterations	Transport Sweeps	Source Iterations
URRd-H_2Ob(1)-2-0-ISLC	92573	14116	52477	6
URRd-H_2Oc(1)-2-0-ISLC	15034	1921	12820	10

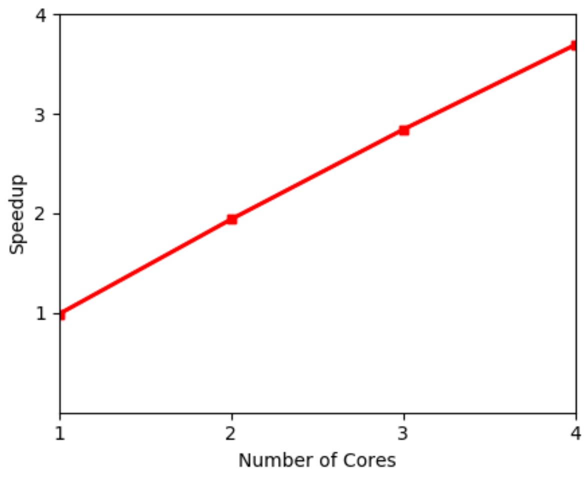
in a 64-bit operating system. Table 4.12 shows the speedup and efficiency of the code obtained using URRd-H_2Ob(1)-2-0-ISLC benchmark with 48 azimuthal divisions and 50 mesh divisions in each region. Figure 4.4a and Figure 4.4b show the variation of speedup and efficiency with increase in number of processor unit.

From Figure 4.4a, we can see that the speed up increases significantly with addition of each processor. However, efficiency also drops by a small few percentages for each addition, as evident from Figure 4.4b. In overall, the efficiency of the scheme is very high and satisfactory, and so, the code for 2D implementation was parallelized in a similar manner.

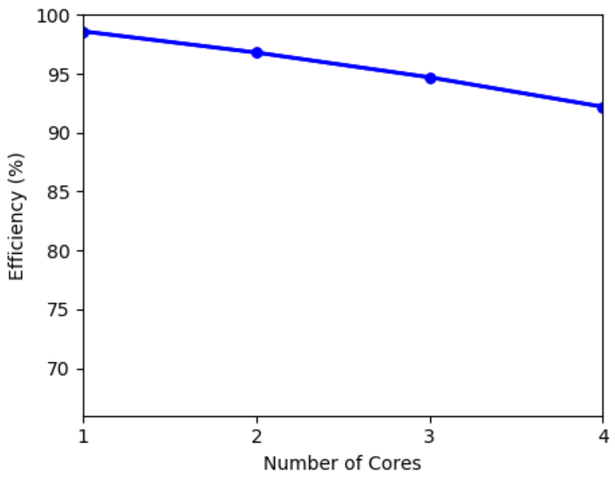
This completes the analysis performed on one-dimensional code for MOC implementation. It provided a general understanding on how the implementation worked subjected to the approximations made and gave a valuable insight on how it would fare in two dimensions. The accuracy of the result is highly dependent on two factors: number of azimuthal divisions and mesh divisions. The accuracy becomes more sensitive to azimuthal divisions in cases with considerable leakage and small geometries, and to mesh divisions when there is high flux gradient. The implementation is capable of decoupling group fluxes very accurately, and is capable of handling heterogeneity with good reliability when adequate mesh numbers are used. The iterative schemes appropriate for solving one-group problems and multigroup problems were also identified and likewise used in two dimensions. The parallelization scheme, developed based on 1D implementation showed satisfactory performance and so it was adopted for 2D implementation.

Table 4.12: Performance Analysis of the Parallelization of the Code for MOC Implementation in One Dimension.

Execution Type	Runtime (s)	Speedup	Efficiency (%)
Serial Execution	734.7	-	-
Parallel Execution:			
1 core	744.9	0.99	98.6
2 core	379.5	1.94	96.8
3 core	258.6	2.84	94.7
4 core	199.2	3.69	92.2



(a) Graph of Speedup against Number of Cores



(b) Graph of Efficiency against Number of Cores

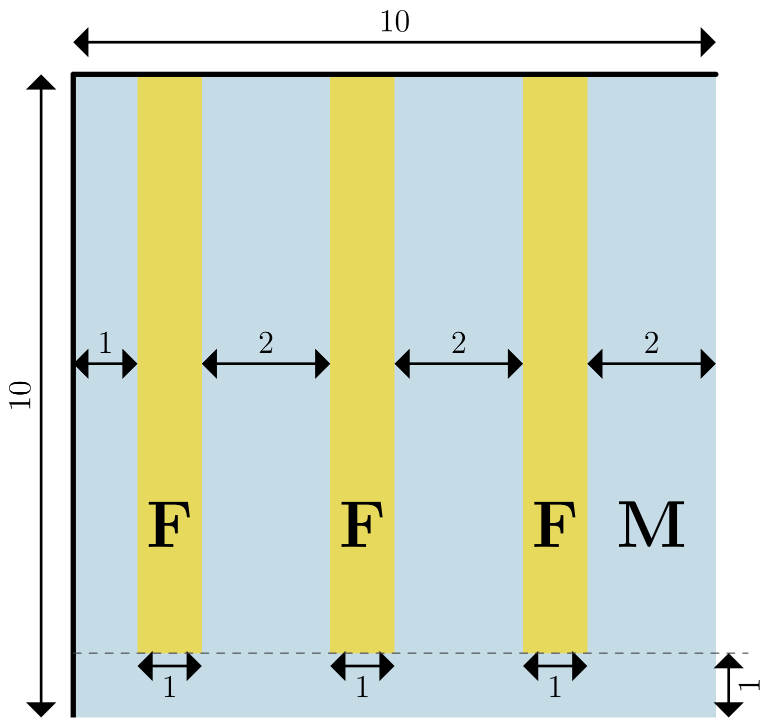
Figure 4.4: Performance Metrics of Parallelization

4.3 One-Group Eigenvalue Solver in Two Dimensions

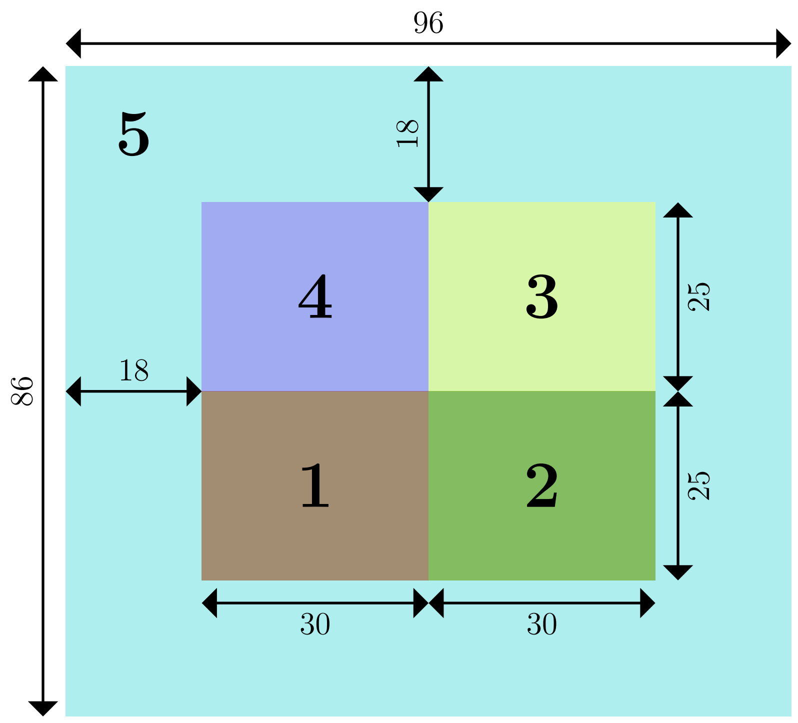
Two benchmark problems were chosen to test the code for one-group solver in two dimensions. The first benchmark, One-Group Eigenvalue Problem, taken from a journal article by Brantley & Larsen, consists of two regions : fuel labeled as 'F' and moderator, labeled as 'M'. The geometry of the problem is shown in Figure 4.5a and the cross section data are provided in Table 4.13. Reflective boundary conditions apply on top and on the left side, and vacuum boundary conditions on the other two sides. The second benchmark, LWR Pool Reactor, was taken from a journal article by J. Stepanek [38]. It consists of five different regions labeled 1,2,3,4 and 5, of which the first four regions are stacked together in the center. The geometry is defined in Figure 4.5b and cross section data are given in Table 4.13 as well. Vacuum boundary conditions apply on all four sides.

The first benchmark was solved using 40 azimuthal divisions, 10000 mesh divisions and ray separation of 0.05 cm. The second benchmark was solved using 32 azimuthal divisions, 8256 mesh divisions and ray separation of 0.5 cm. The results are shown in Table 4.14 along with comparisons of results from other codes. The flux distributions are shown in Figure 4.6. The results show a satisfactory agreement for the first benchmark problem but deviates slightly for the second benchmark. This is because of the high flux gradients as can be observed in the graph in region 1, which adds some error due to flat source approximation.

The flux distribution for One-Group Eigenvalue Problem was also compared with the distributions from other methods: TWODANT S_{16}, SP_3 and SP_1, at y=5.5 cm and y=1.033 cm as shown in Figure 4.7. The distribution found follows the same pattern as in the TWODANT code but is a bit depressed at y=5.5 cm and has some discrepancies at y=1.033 cm in the fuel region, where the flux peaks a bit more, as can be seen in the figures. This indicates that the distribution from the solver is flatter along y-direction and is more curvy along x-direction near the edge than that of TWODANT. Nevertheless, it can be seen that the solver is a far more powerful tool in resolving a finer flux distribution than SP_1 method or SP_3 method.



(a) One-Group Eigenvalue Problem



(b) LWR Pool Reactor

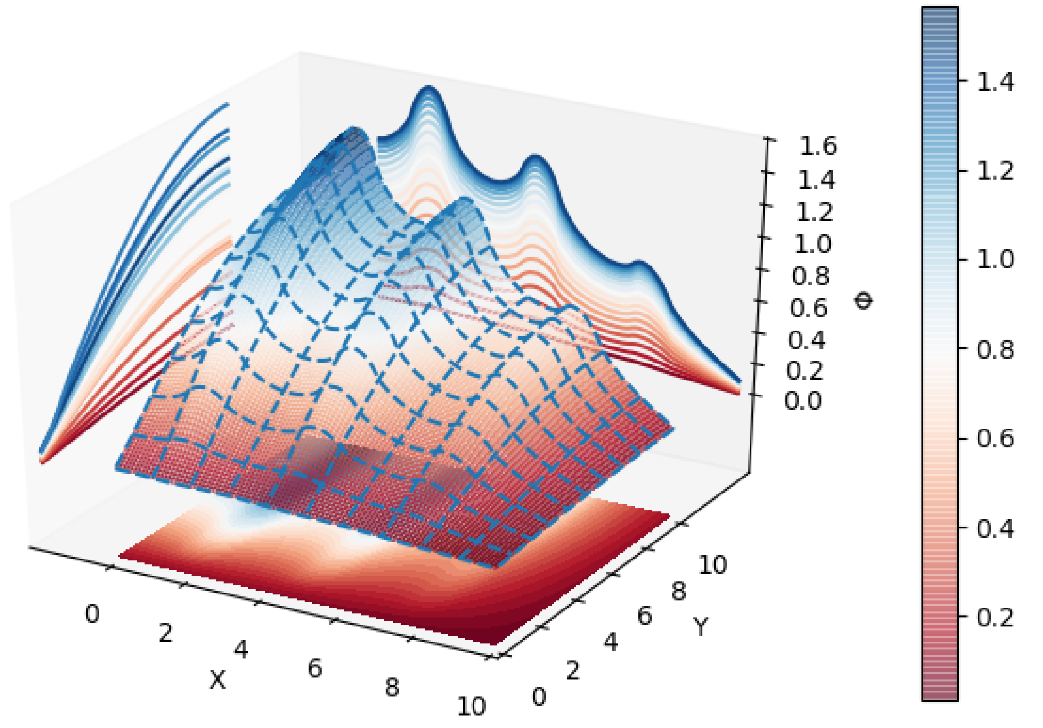
Figure 4.5: Geometries of Benchmark Problems for One-Group Solver in Two Dimensions(all dimensions are in cm)

Table 4.13: Cross Section Data for One Group Problem Sets in Two Dimensions

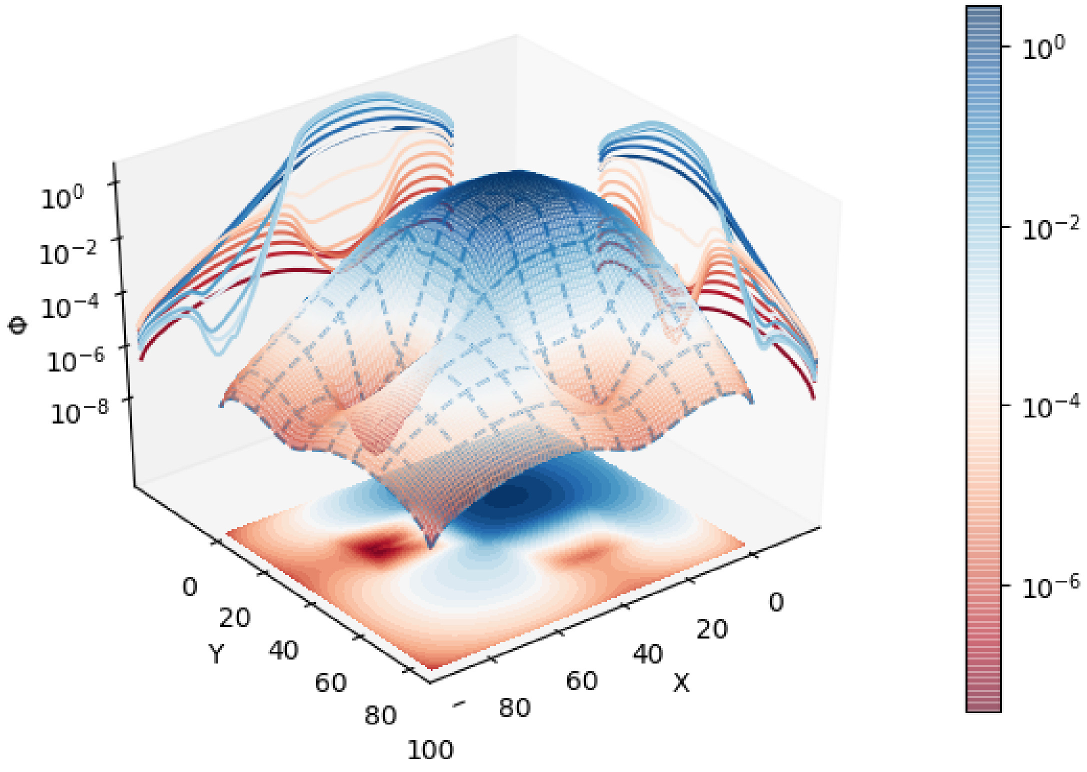
Problem		Σ_t (cm^{-1})	Σ_s (cm^{-1})	$\nu\Sigma_f$ (cm^{-1})
One-Group Eigenvalue Problem:	M	1.0	0.93	0.0
	F	1.5	1.35	0.24
LWR Pool Reactor:	1	0.60	0.53	0.079
	2	0.48	0.20	0.0
	3	0.70	0.66	0.043
	4	0.65	0.50	0.0
	5	0.90	0.89	0.0

Table 4.14: Comparison of Results for One-Group Benchmark Problems in Two-Dimension with Other Codes

Problem		k_{eff}	Difference
One-Group Eigenvalue Problem	Python MOC Solver	0.805367	
	TWODANT [39]	0.806132	7.67×10^{-4}
	TEPFEM [40]	0.803068	2.30×10^{-3}
	TPTRI [41]	0.806123	7.58×10^{-4}
	RRSD & RPNES [42]	0.806464	1.10×10^{-3}
LWR Pool Reactor	Python MOC Solver	1.0209	
	SURCU [38]	1.0069	0.0140
	FELICIT [43]	1.0069	0.0140
	TEPFEM	1.0079	0.0130
	TPTRI	1.0070	0.0139
	RRSD & RPNES	1.0044	0.0165

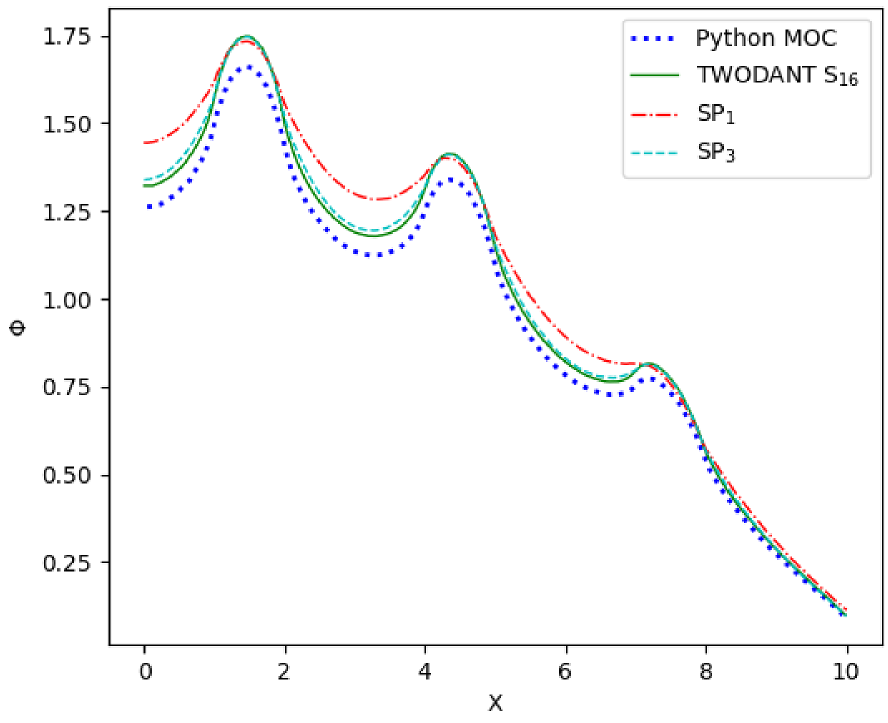


(a) One-Group Eigenvalue Problem

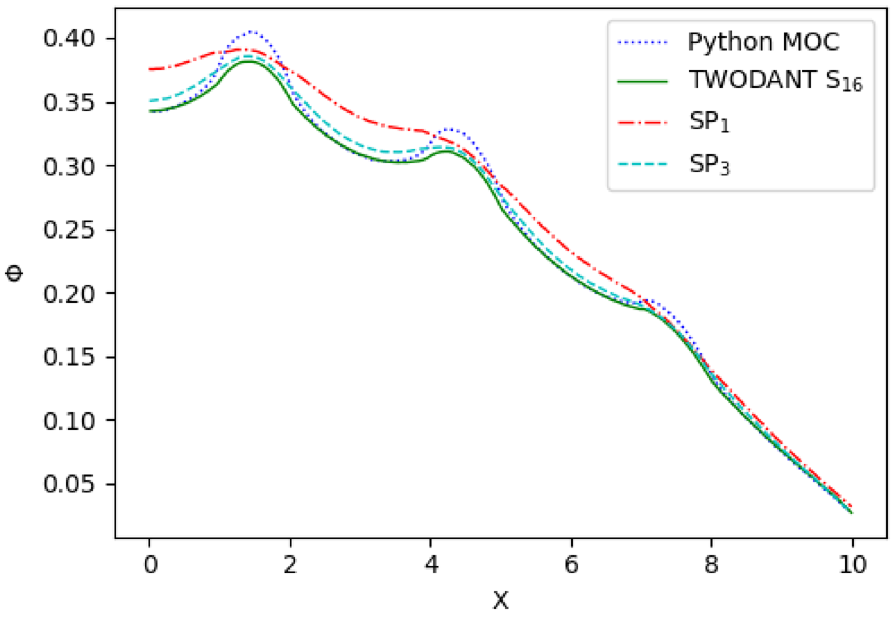


(b) LWR Pool Reactor

Figure 4.6: Eigenfunctions of Benchmark Problems obtained from One-Group Solver in Two Dimensions



(a) y=5.5 cm



(b) y=1.033 cm

Figure 4.7: Comparison of Flux Distributions for One-Group Eigenvalue Problem

4.4 Multigroup Eigenvalue Solver in Two Dimensions

At this final phase of the development of MOC code, the multigroup solver was first tested on a benchmark problem taken from [38]. Its a BWR Cell which consist of two mediums, a homogenized fuel element surrounded by light water. The geometry of the problem is shown in Figure 4.8 and the cross section data are provided in Table 4.15. Fully reflective boundary conditions apply on all four sides. The problem involves upscattering, as can be seen from cross-section data.

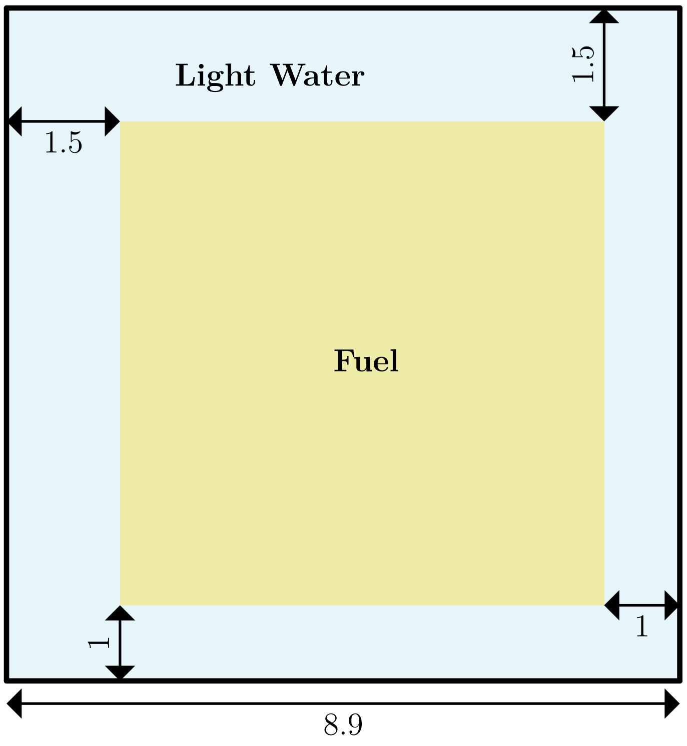


Figure 4.8: Geometry of BWR Cell Problem (all dimensions are in cm)

Table 4.15: Cross Section Data for BWR Cell Problem

Material	g	χ_g	$\nu\Sigma_f^g$	Σ_t^g	$\Sigma_s^{1\to g}$	$\Sigma_s^{2\to g}$
Fuel	1	1	6.203×10^{-3}	1.96647×10^{-1}	1.78×10^{-1}	1.089×10^{-3}
	2	0	1.101×10^{-1}	5.96159×10^{-1}	1.002×10^{-2}	5.255×10^{-1}
Light Water	1	0	0.0	2.22064×10^{-1}	1.995×10^{-1}	1.558×10^{-3}
	2	0	0.0	8.87874×10^{-1}	2.118×10^{-2}	8.783×10^{-1}

The benchmark was solved using 31684 meshes and 40 azimuthal divisions and ray separation of 0.05 cm. The results are shown in Table 4.16. The eigenvalue is slightly off by around 100 pcm from other calculated values. Figure 4.9 shows the flux profiles of the fast and thermal neutrons obtained from the solver. The fast flux produced in the middle by fuel gets moderated in the periphery by light water, so the profile is concave down, as expected. The thermal flux produced by moderation at the periphery get absorbed in the middle by the fuel to perform fission. This high absorption leads to self-shielding, so the thermal flux profile is concave up, as expected. The accuracy of the eigenvalue and the consistency of the flux profiles with the understandings of reactor physics demonstrate the capability of the solver for handling multigroup problems in two dimensions.

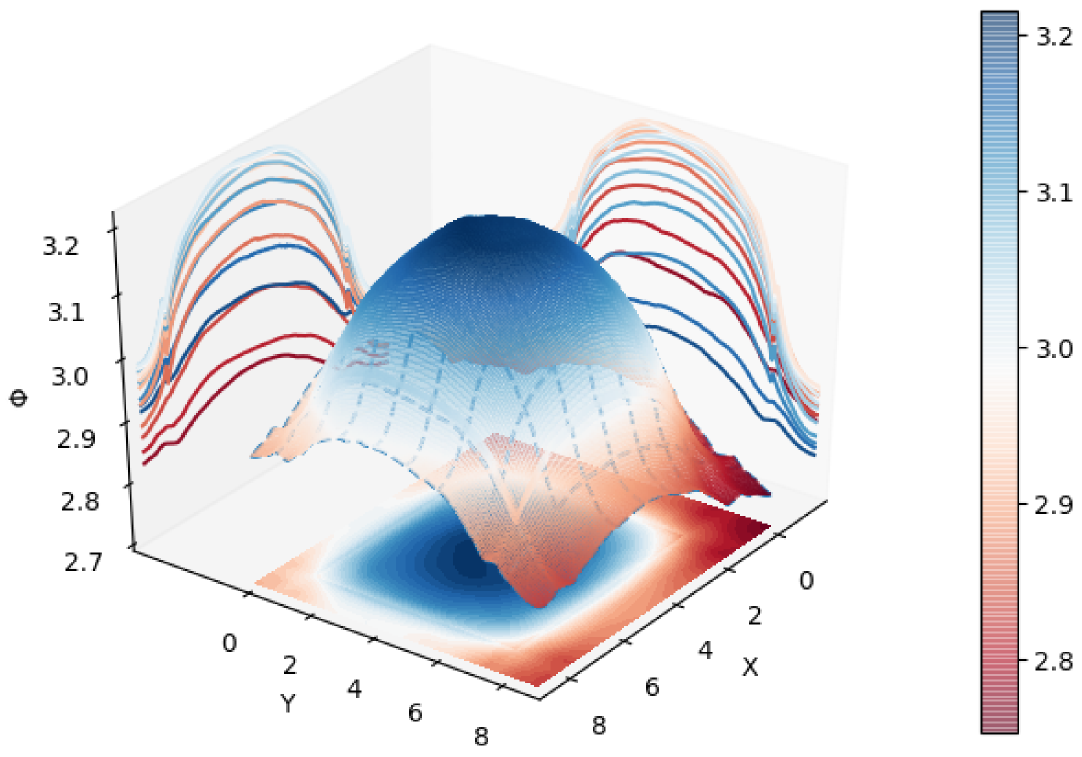
Table 4.16: Comparison of Results for BWR Cell Problem with Other Codes

	k	Difference
Pyhton MOC Solver	1.2114	
SURCU	1.2127	1.3×10^{-3}
TEPFEM	1.2136	2.2×10^{-3}
TPTRI	1.2128	1.4×10^{-3}
RRSD & RPNES	1.2119	5×10^{-4}

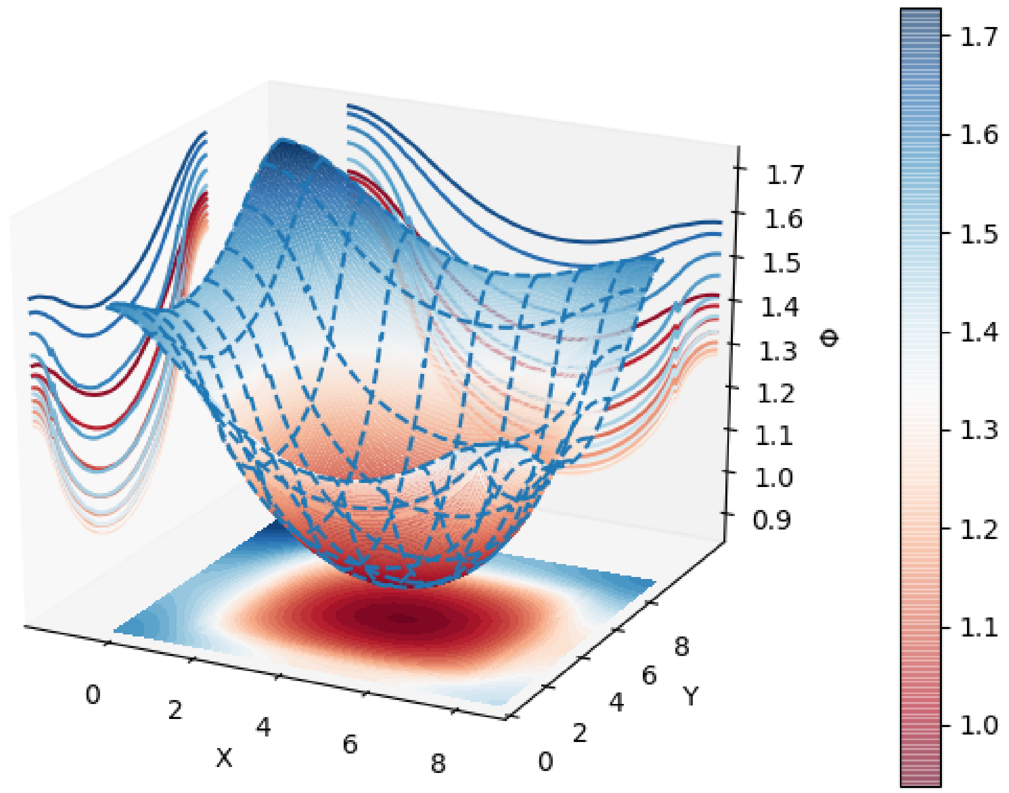
The solver is now applied to test its performance in resolving neutronics at assembly-level calculations. Four benchmark problems selected for this purpose were taken from [44]. The benchmarks consist of three square-shaped assemblies: UX assembly, UA assembly and PX assembly. Each assembly consists of 289 square cells, with each cell 1.26 cm in length.

The arrangement of these assemblies are shown in Figures 4.10. The cross-section data for each of the cells in these assemblies are provided in Table 4.17. The core configurations for each of the four benchmark problems, identified as C1, C2, C3 and C4V are described in Table 4.18.

The C1 configuration consists of UX assemblies and UA assemblies in infinite checker board set-up, as can be seen in the table. This means periodic boundary conditions apply on all sides of the outlined section. This presents an inconvenience for the solver as it is not made capable of handling periodic boundary conditions yet. However, given the symmetry of the configuration and the assemblies, there is a work-around for this inconvenience. The boundary of the geometry was set on the lines connecting the centers of the adjacent assemblies and applying



(a) Fast Flux Distribution



(b) Thermal Flux Distribution

Figure 4.9: Eigenfunctions for BWR Cell Problem

Table 4.17: Cross Section Data for NEA Benchmark Problems

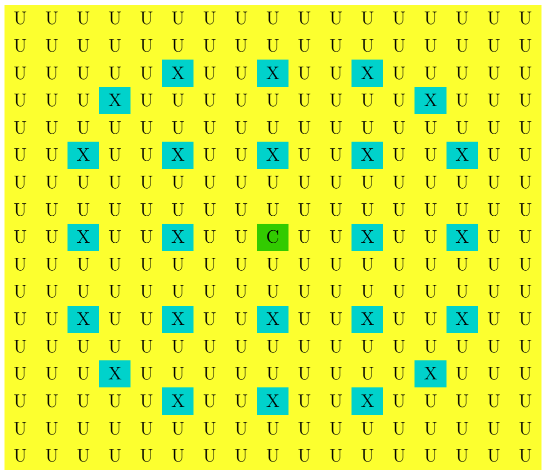
Cell Type			g	χ_g	$\nu\Sigma_f^g$	Σ_a^g	$\Sigma_s^{1\rightarrow g}$	$\Sigma_s^{2\rightarrow g}$	μ_g
U	:	UO_2 Fuel	1	1	0.0050	0.010	0.54	0	0.50
			2	0	0.125	0.100	0.020	1.00	0.30
P_1	:	Peripheral MOX Fuel	1	1	0.0075	0.015	0.52	0	0.50
			2	0	0.300	0.200	0.015	0.90	0.30
P_2	:	Intermediate MOX Fuel	1	1	0.0075	0.015	0.52	0	0.50
			2	0	0.375	0.250	0.015	0.83	0.30
P_3	:	Central MOX Fuel	1	1	0.0075	0.015	0.52	0	0.50
			2	0	0.450	0.300	0.015	0.76	0.30
X	:	Guide Tube	1	0	0	0.001	0.56	0	0.50
			2	0	0	0.02	0.025	1.20	0.30
R	:	Reflector	1	0	0	0.001	0.56	0	0.50
			2	0	0	0.04	0.050	2.30	0.30
C	:	Moveable Fission Chamber	1	1	1×10^{-7}	0.001	0.56	0	0.50
			2	0	3×10^{-6}	0.02	0.025	1.20	0.30
A	:	Absorber (AIC)	1	0	0	0.040	0.48	0	0.50
			2	0	0	0.8	0.010	0.05	0.30

reflective boundary conditions on all sides. This way the total area of the geometry is reduced by 4 times, thus reducing the computational load by 4 times as well.

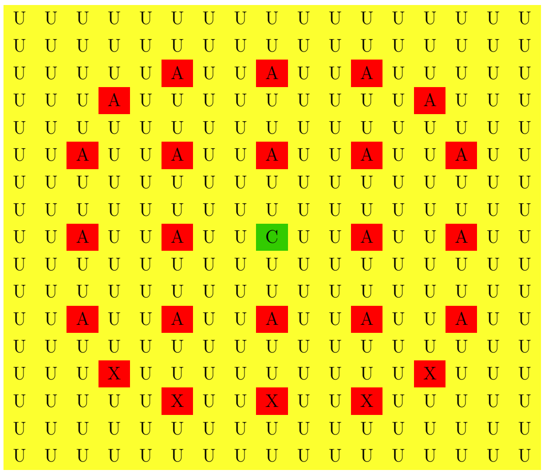
The C2 configuration is similar to C1 except the UA assemblies were replaced by PX assemblies. Hence this benchmark was also solved in a similar manner to C1 configuration. The C3 configuration consists of two UX and PX assemblies in checker board configuration with reflective boundary conditions on all sides. C4V configuration is similar to C3 except that vacuum boundary conditions apply at the right and at the bottom.

The algorithm is incapable of handling anisotropic scattering since all scattering was assumed to be isotropic. So P_0 approximations were applied, i.e. the cross-sections were corrected for transport. The transport-corrected self-scattering cross-section, $\Sigma_{s,tr}^{g\rightarrow g}$, was calculated:

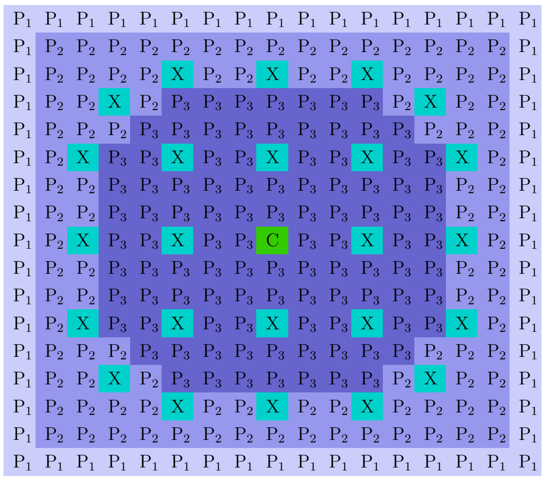
$$\Sigma_{s,tr}^{g\rightarrow g} = \Sigma_s^{g\rightarrow g}\left(1-\mu_g\right). \tag{4.1}$$


U U U U U U U U U U U U U U U U U
U U U U U U U U U U U U U U U U U
U U U U U X U U X U U X U U U U U
U U U X U U U U U U U U X U U U U
U U U U U U U U U U U U U U U U U
U U X U U X U U X U U X U U X U U
U U U U U U U U U U U U U U U U U
U U U U U U U U U U U U U U U U U
U U X U U X U U C U U X U U X U U
U U U U U U U U U U U U U U U U U
U U U U U U U U U U U U U U U U U
U U X U U X U U X U U X U U X U U
U U U U U U U U U U U U U U U U U
U U U X U U U U U U U U X U U U U
U U U U U X U U X U U X U U U U U
U U U U U U U U U U U U U U U U U
U U U U U U U U U U U U U U U U U

(a) UX Assembly


U U U U U U U U U U U U U U U U U
U U U U U U U U U U U U U U U U U
U U U U U A U U A U U A U U U U U
U U U A U U U U U U U U A U U U U
U U U U U U U U U U U U U U U U U
U U A U U A U U A U U A U U A U U
U U U U U U U U U U U U U U U U U
U U U U U U U U U U U U U U U U U
U U A U U A U U C U U A U U A U U
U U U U U U U U U U U U U U U U U
U U U U U U U U U U U U U U U U U
U U A U U A U U A U U A U U A U U
U U U U U U U U U U U U U U U U U
U U U X U U U U U U U U X U U U U
U U U U U X U U X U U X U U U U U
U U U U U U U U U U U U U U U U U
U U U U U U U U U U U U U U U U U

(b) UA Assembly


P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1
P_1 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_1
P_1 P_2 P_2 P_2 P_2 X P_2 P_2 X P_2 P_2 X P_2 P_2 P_2 P_2 P_1
P_1 P_2 P_2 X P_2 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_2 X P_2 P_2 P_1
P_1 P_2 P_2 P_2 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_2 P_2 P_2 P_1
P_1 P_2 X P_3 P_3 X P_3 P_3 X P_3 P_3 X P_3 P_3 X P_2 P_1
P_1 P_2 P_2 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_2 P_2 P_1
P_1 P_2 P_2 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_2 P_2 P_1
P_1 P_2 X P_3 P_3 X P_3 P_3 C P_3 P_3 X P_3 P_3 X P_2 P_1
P_1 P_2 P_2 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_2 P_2 P_1
P_1 P_2 P_2 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_2 P_2 P_1
P_1 P_2 X P_3 P_3 X P_3 P_3 X P_3 P_3 X P_3 P_3 X P_2 P_1
P_1 P_2 P_2 P_2 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_2 P_2 P_2 P_1
P_1 P_2 P_2 X P_2 P_3 P_3 P_3 P_3 P_3 P_3 P_3 P_2 X P_2 P_2 P_1
P_1 P_2 P_2 P_2 P_2 X P_2 P_2 X P_2 P_2 X P_2 P_2 P_2 P_2 P_1
P_1 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_2 P_1
P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1 P_1

(c) PX Assembly

Figure 4.10: Assembly Arrangements for NEA Benchmark Problems

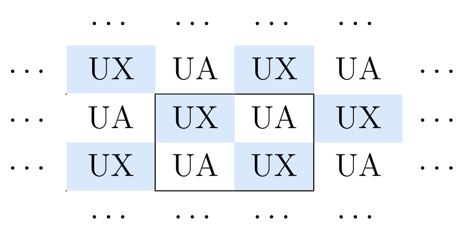
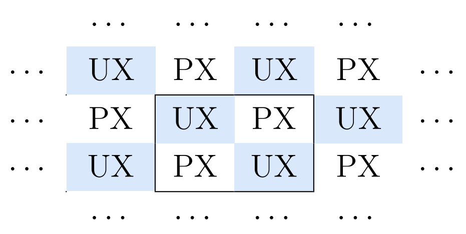
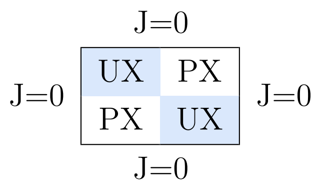
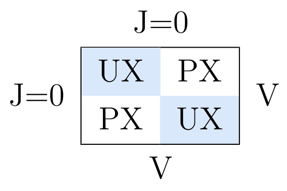
The transport-corrected total cross-section, $\Sigma^g_{t,tr}$ was then calculated:

$$\Sigma^g_{t,tr} = \Sigma^g_a + \Sigma^{g\to g}_{s,tr} + \sum_{g,g\neq g'}^{G} \Sigma^{g\to g'}_s \tag{4.2}$$

All the four benchmarks were solved using 16 meshes for each cell and 32 azimuthal divisions with a ray separation of 0.315 cm. The result for eigenvalues are summarized in Table 4.19. The table compares the eigenvalues with the mean eigenvalues determined from the calculations by different transport codes. The heterogeneous production rate distributions (pin-by-pin fission-rate distribution) were also calculated. The distributions were normalized so that

$$\frac{1}{k}\left[\sum_i^I \sum_g^G \nu\Sigma_f^{g,i}\Phi_{g,i}\right] = 1. \tag{4.3}$$

Table 4.18: Core Configurations for NEA Benchmark Problems

Configuration	Geometric description
C1 Uranium Infinite Checker Board	
C2 MOX Infinite Checker Board	
C3 Reflected MOX Checker Board	 J=0 represents reflective boundary conditions
C4V Semi Reflected MOX Checker Board	 V represents vacuum boundary conditions

These distributions were averaged along the symmetries of the geometry and are elaborately shown and compared with the mean values in Figure 4.13 for C1, Figure 4.15 for C2, Figure 4.17 for C3, and Figure 4.19 for C4V. The results are

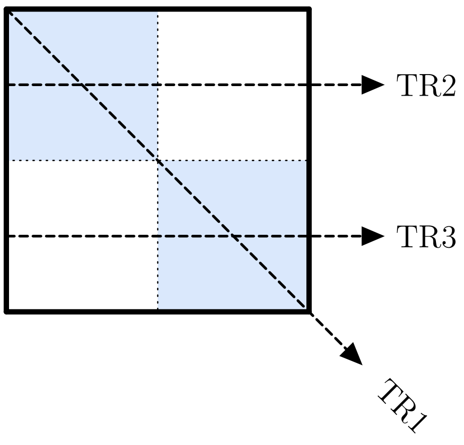


Figure 4.11: Traverses along which error was compared with other transport calculations

summarized in Table 4.20.

The pin power distributions were also compared with other transport calculations using DOT 4.3 (S_8) [45] with diagonal correction by **CEN**; DORT (S_8) in PASC-3 code system [46] by **ECN**; SRAC (Collision Probability) [47], TWOTRAN II (S_4) [48] by **JAERI**; QP1 (Transmission Probability Method) [49] in BOXER by **PSI**; VIM (MC) [50] by **Argonne**; ICM 2D ($J\pm$) [51] in RSYST by **IKE**; TWODANT (S_{16}) [52] by **NFI** as reported in [44] along lines TR1, TR2 (and TR3 for C4V only) as defined in Figure 4.11. The datasets that were difficult to scale were excluded. The plots are shown in Figure 4.14 for C1, Figure 4.16 for C2, Figure 4.18 for C3, and Figure 4.20 for C4V.

For C1, the eigenvalue deviates from mean significantly by around 250 pcm, and is barely within the standard deviation of the mean. For C2 and C3, the eigenvalue-k are in excellent agreement with the mean. The result for C4V shows a huge deviation in terms of eigenvalue, around 500 pcm, which is quite large compared to the standard deviation.

In both C1 and C4V, there is a presence of high flux gradient introduced by absorbing medium in UA assembly at C1 and leakage at C4V. This means that the error introduced by flat source approximation will be quite significant in both cases and finer meshes will be required to obtain a more accurate solution. However, increasing the number of meshes any further does not improve the results.

Upon inspection on effect of increasing mesh divisions, it was seen that the deviation of eigenvalue was not caused by flat source approximation. The likely

Table 4.19: NEA Benchmarks Eigenvalues

Core Configurations	Results from Transport Codes			k calculated by Python MOC Solver	Difference $\bar{k} - k$
	Number of Calculations	Mean Eigenvalue-$\bar{k}$	Standard Deviation		
C1	13	0.84794	0.00270	0.85045	0.00251
C2	12	1.02287	0.00043	1.02279	0.00008
C3	10	1.01795	0.00048	1.01789	0.00006
C4V	8	0.91720	0.00079	0.92208	0.00488

cause of such discrepancy is inadequate modeling of anistropicity. P_0 approximation works well in in low absorbing medium in thermal reactors but in presence of absorbers, a higher order approximation is necessary.

For C1, in UX assembly, the pin-power error is relatively low except along the edge, where the negative flux gradient starts picking up. The r.m.s. error for the assemblies is thus much lower (0.62%) with maximum error about 1.18%. The eigenfunction in this region is well resolved, and is far better than most transport codes such that, numerical errors becomes apparent, as can be seen in Figure 4.14a, where some points are above the zero line and some points are below the zero line.

In UA assembly, the anistropicity introduces significant error in the vicinity of Absorber cells. The error is positive and about 5-7% for cells within the Absorber cells, with an r.m.s. error of about 4.05% for the whole assembly, and is the cause of the error in the eigenvalue observed. In both assemblies, the error is positive through out. In Figure 4.14b, it can be seen that such errors are present in almost all other transport calculations. The MOC Solver performs within an acceptable level in this regard.

In C2 and C3, although the eigenvalues are very close to the referenced mean values from different calculations, the eigenfunctions present an appreciable discrepancy. In both configurations, the power is a bit elevated in the PX assemblies with r.m.s. errors of 0.86% and 1.40% and depressed in UX assemblies with r.m.s. errors of 0.97% and 1.69% for C2 and C3 configurations respectively. These fluctuations are more pronounced in C3 than C2 with r.m.s. error increasing by 0.54% for PX assemblies and by 0.72% for UX assemblies from C2 to C3. This is because reflective boundary condition means four UX or PX assemblies stacked together alternately rather than just one in C2, allowing the error to amplify further. However, these alternate errors cancels out, and so the eigenvalue does not show much deviation.

In Figure 4.16 and in Figure 4.18, it can be seen that such pattern of error occurs in all the transport calculations, but the errors are much less amplified than the MOC Solver. It is speculated that a linear treatment of anisotropicity in MOC solver will significantly dampen the error for MOX combinations [53].

In C4V, the vacuum interface induces some negative error in its vicinity. In top left UX assembly, most of the pin power errors is positive and is observed along the periphery, and is minimal in the center, as can be seen in Figure 4.19a. For PX assemblies, the error peaks near the top-left UX-PX interface along the Guide Tube cells, and then again decreases and becomes negative near the vacuum interface. The absolute error in bottom right UX assembly is minimal, with r.m.s. error about one-third to that of C3 configuration.

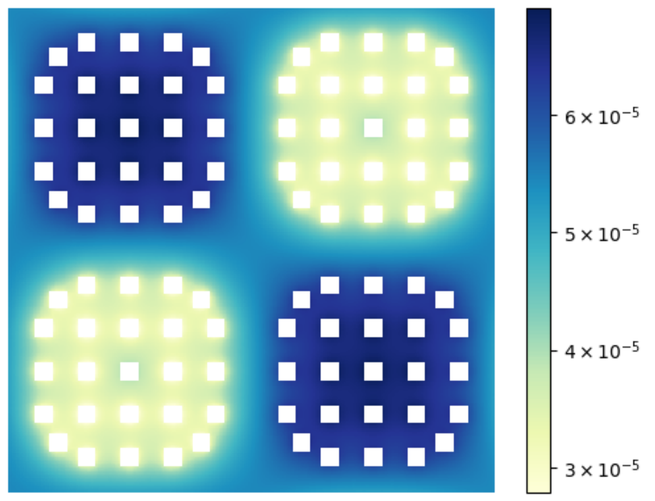
The vacuum condition dampens the oscillatory errors in C4V as observed in C2 and C3, strongly in UX assemblies and slightly in PX assemblies. Thus the errors in PX assemblies is not negated by the errors in UX assemblies, which in turn, lead to a discrepancy of 500 pcm in the eigenvalue.

Another way to realize this error is to see that anisotropicity also introduces error in leakage situations [54]. This error in bottom right UX assembly canceled out the error induced by the adjacent PX assemblies, and thus the error in eigenvalue originates from both PX assemblies and vacuum conditions due to inadequate modeling of the anisotropicity in the algorithm.

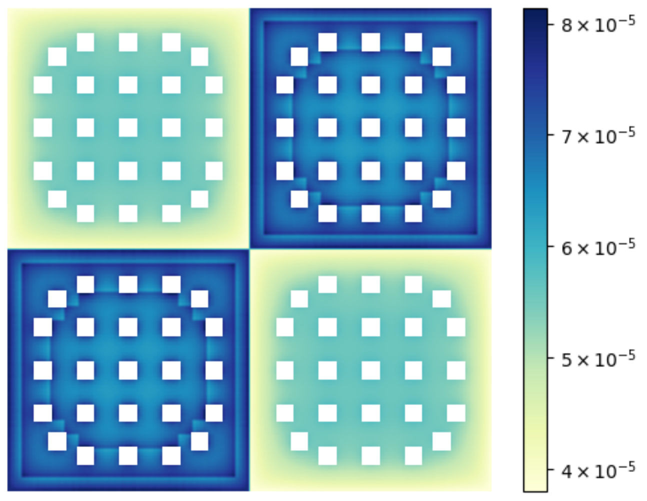
In Figure 4.20, it can be seen that the error in pins of UX assemblies is comparable to that from other transport calculations. However, the error in PX assembly pins are still much higher, as observed in C3. In the vicinity of vacuum, the transport calculations, with which the result was compared, also have negative error, similar to the MOC Solver. It is speculated that a P_1 treatment of scattering anisotropicity will allow this code to perform calculations with at least as much accuracy as any of these transport calculations with which the results were compared.

Table 4.20: Summary of NEA Benchmark Results for Heterogeneous Production Rate Distribution Errors

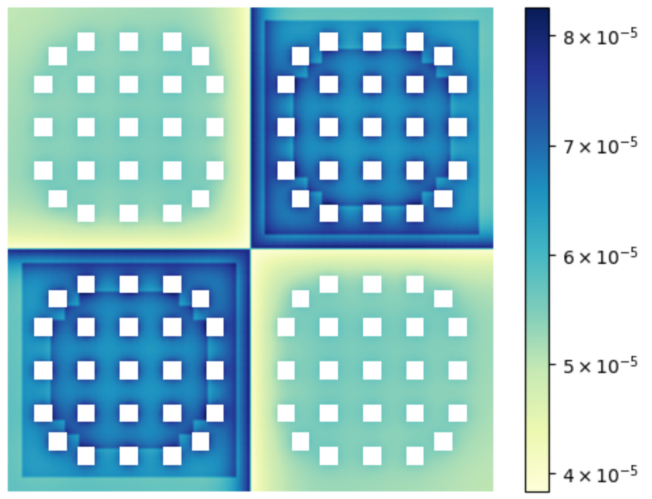
	Core Configurations	Peak Pin Power Relative Error (ave. %)	Pin Power Relative Error (%)		Pin Power Absolute Error($\times 10^{-6}$)	
			Maximum	R.m.s	Maximum	R.m.s
C1:	UX Assembly	0.37	1.18	0.62	10	5.53
	UA Assembly	1.05	7.35	4.05	36	22.4
	Overall	0.37	7.35	2.90	36	16.3
C2:	UX Assembly	-1.09	1.94	0.97	14	8.18
	PX Assembly	-1.31	1.68	0.86	18	9.57
	Overall	-1.31	1.94	0.92	18	8.90
C3:	UX Assembly	-1.64	2.90	1.69	26	14.8
	PX Assembly	-1.13	3.06	1.40	27	14.4
	Overall	-1.13	3.06	1.55	27	14.6
C4V:	UX Assembly (top left)	0.29	2.76	0.84	26	9.18
	PX Assembly	0.17	7.24	2.00	41	12.5
	UX Assembly (bottom right)	0.14	10.0	2.86	16	4.19
	Overall	0.29	10.0	2.05	41	10.2



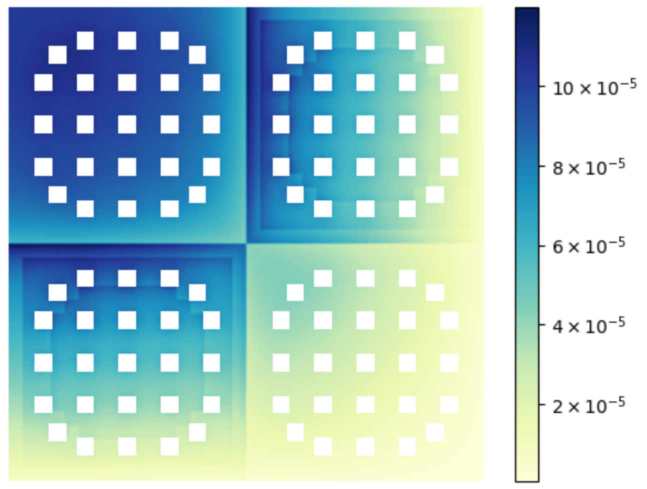
(a) C1 Configuration



(b) C2 Configuration



(c) C3 Configuration



(d) C4V Configuration

Figure 4.12: Fission Rate Distributions for NEA Benchmark Results

Key
Calculated
Reference
Difference

								872 864 0.93%
							890 883 0.79%	874 866 0.92%
						937 932 0.54%	904 898 0.67%	873 865 0.92%
					-	970 965 0.52%	915 910 0.55%	870 861 1.05%
				1026 1026 0.00%	1019 1016 0.30%	988 984 0.41%	927 923 0.43%	866 858 0.93%
			-	1040 1037 0.29%	1020 1018 0.20%	-	940 934 0.64%	863 854 1.05%
		1054 1053 0.09%	1058 1054 0.38%	1031 1030 0.10%	1008 1007 0.10%	989 984 0.51%	929 924 0.54%	860 851 1.06%
	1065 1064 0.09%	1059 1058 0.09%	1062 1059 0.28%	1033 1032 0.10%	1010 1008 0.20%	989 984 0.51%	928 924 0.43%	858 849 1.06%
-	1080 1076 0.37%	1074 1070 0.37%	-	1047 1044 0.29%	1022 1018 0.39%	-	939 932 0.75%	858 850 0.94%

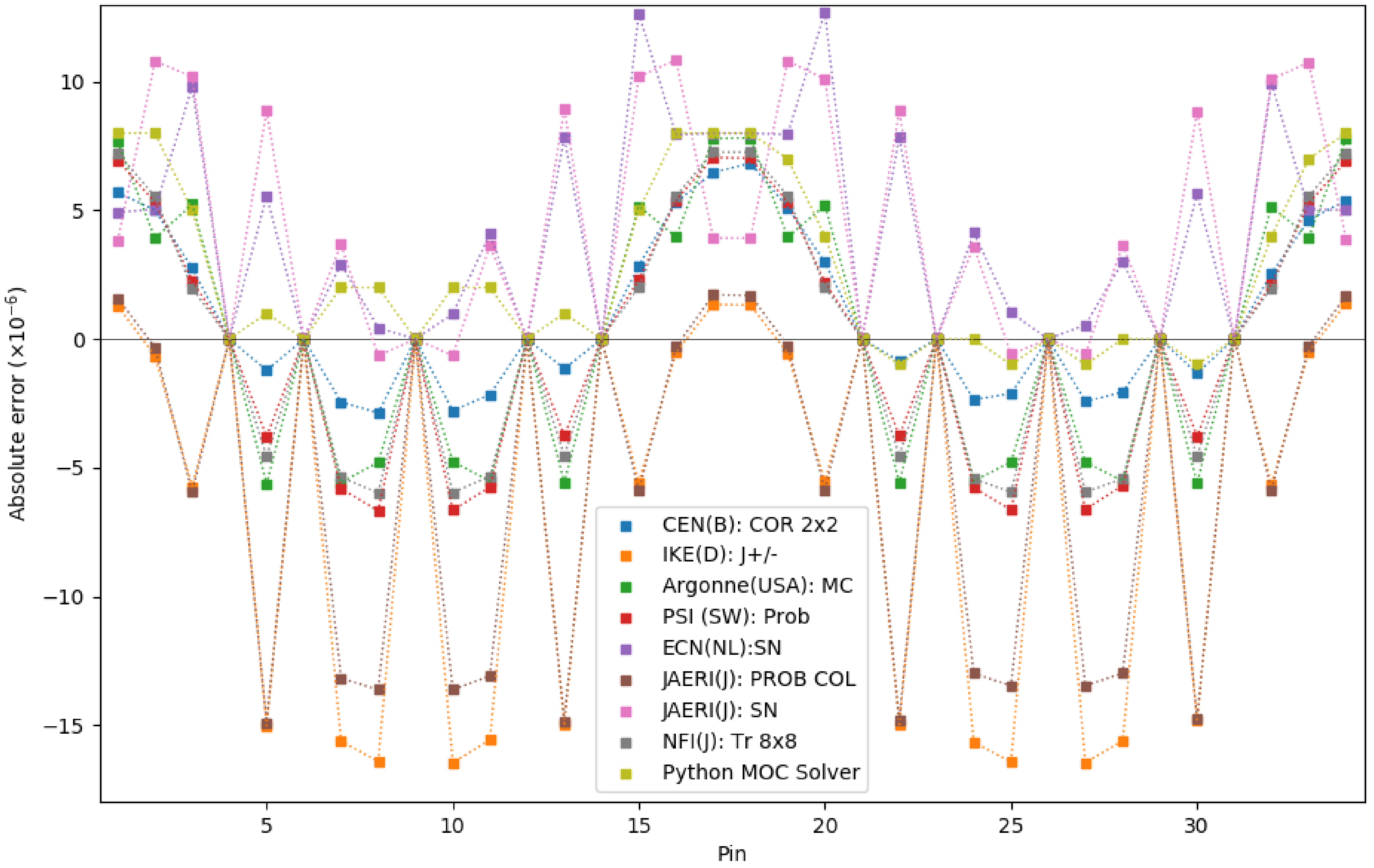
(a) UX Assembly

Key
Calculated
Reference
Difference

								865 856 1.05%
							824 811 1.60%	853 843 1.19%
						711 689 3.19%	787 770 2.21%	836 825 1.33%
					-	623 598 4.18%	745 724 2.90%	817 804 1.62%
				524 490 6.94%	517 487 6.16%	570 543 4.97%	696 674 3.26%	797 782 1.92%
			-	509 480 6.04%	524 495 5.86%	-	647 628 3.03%	783 768 1.95%
		540 513 5.26%	505 481 4.99%	543 514 5.64%	563 533 5.63%	568 546 4.03%	675 652 3.53%	780 765 1.96%
	588 569 3.34%	550 526 4.56%	506 484 4.55%	544 516 5.43%	563 535 5.23%	568 546 4.03%	674 650 3.69%	778 763 1.97%
-	603 584 3.25%	523 507 3.16%	-	514 489 5.11%	530 505 4.95%	-	641 621 3.22%	775 759 2.11%

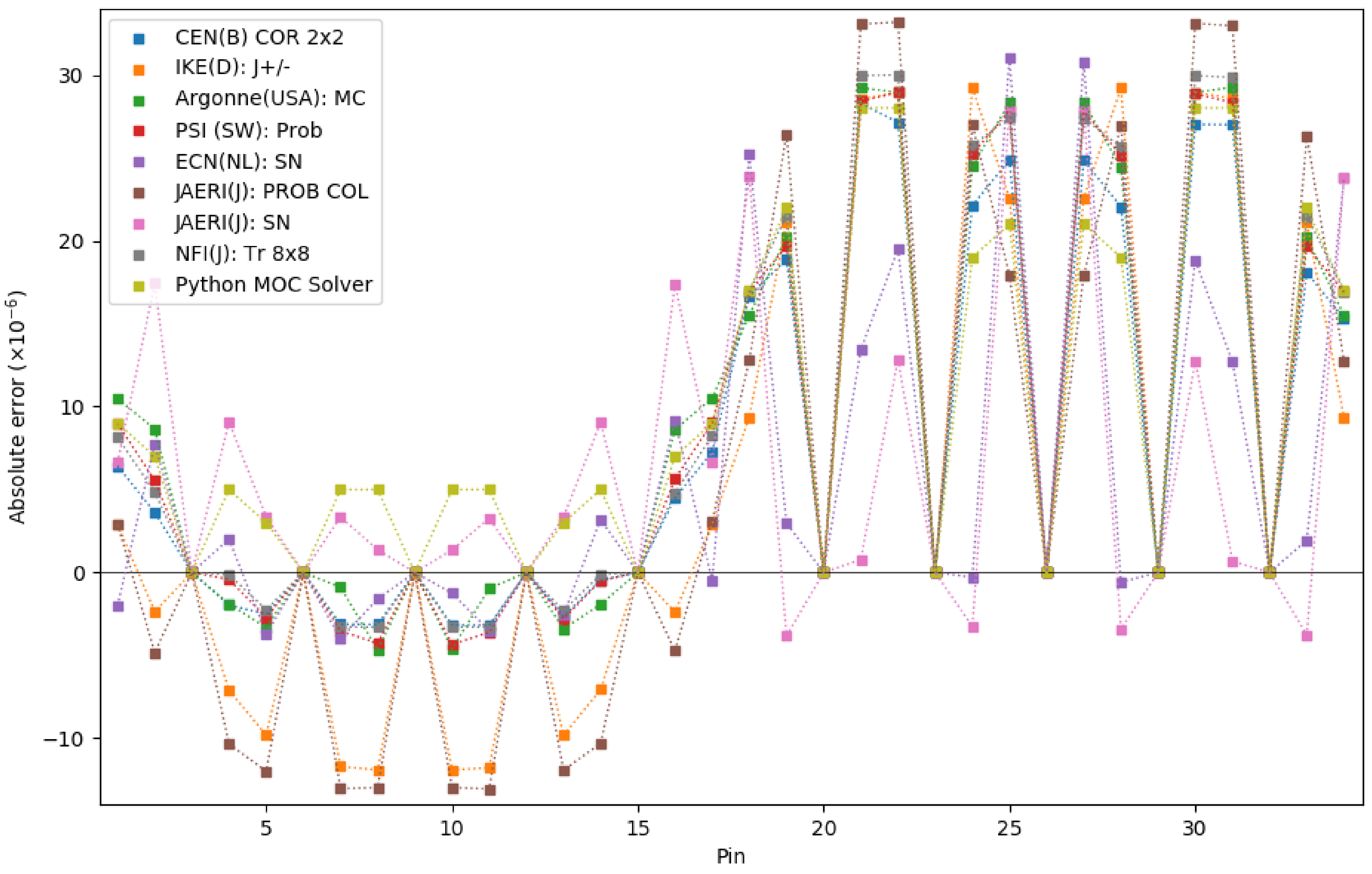
(b) UA Assembly

Figure 4.13: Comparison of Heterogeneous Production Rate Distribution ($\times 10^{-6}$) for C1 Configuration



(a) Absolute Error along TR1

Figure 4.14: Comparison of Pin Power Error for C1 Configuration with other transport calculations



(b) Absolute Error along TR2

Figure 4.14: Comparison of Pin Power Error for C1 Configuration with other transport calculations (cont.)

Key
Calculated Reference Difference

								630 618 1.94%
							719 715 0.56%	657 648 1.39%
						799 804 -0.62%	750 750 0.00%	670 663 1.06%
					-	837 843 -0.71%	770 772 -0.26%	680 674 0.89%
				890 902 -1.33	886 896 -1.12	856 863 -0.81%	787 790 -0.38%	687 682 0.73%
			-	898 909 -1.21%	885 895 -1.12%	-	802 803 -0.12%	692 687 0.73%
		895 907 -1.32%	904 914 -1.09%	886 898 -1.34%	872 884 -1.36%	856 863 -0.81%	794 796 -0.25%	693 687 0.87%
	897 909 -1.32%	896 908 -1.32%	904 914 -1.09%	886 898 -1.34%	872 884 -1.36%	856 863 -0.81%	794 796 -0.25%	693 687 0.87%
-	909 919 -1.09%	908 917 -0.98%	-	897 907 -1.10%	883 892 -1.01%	-	803 805 -0.25%	693 689 0.58%

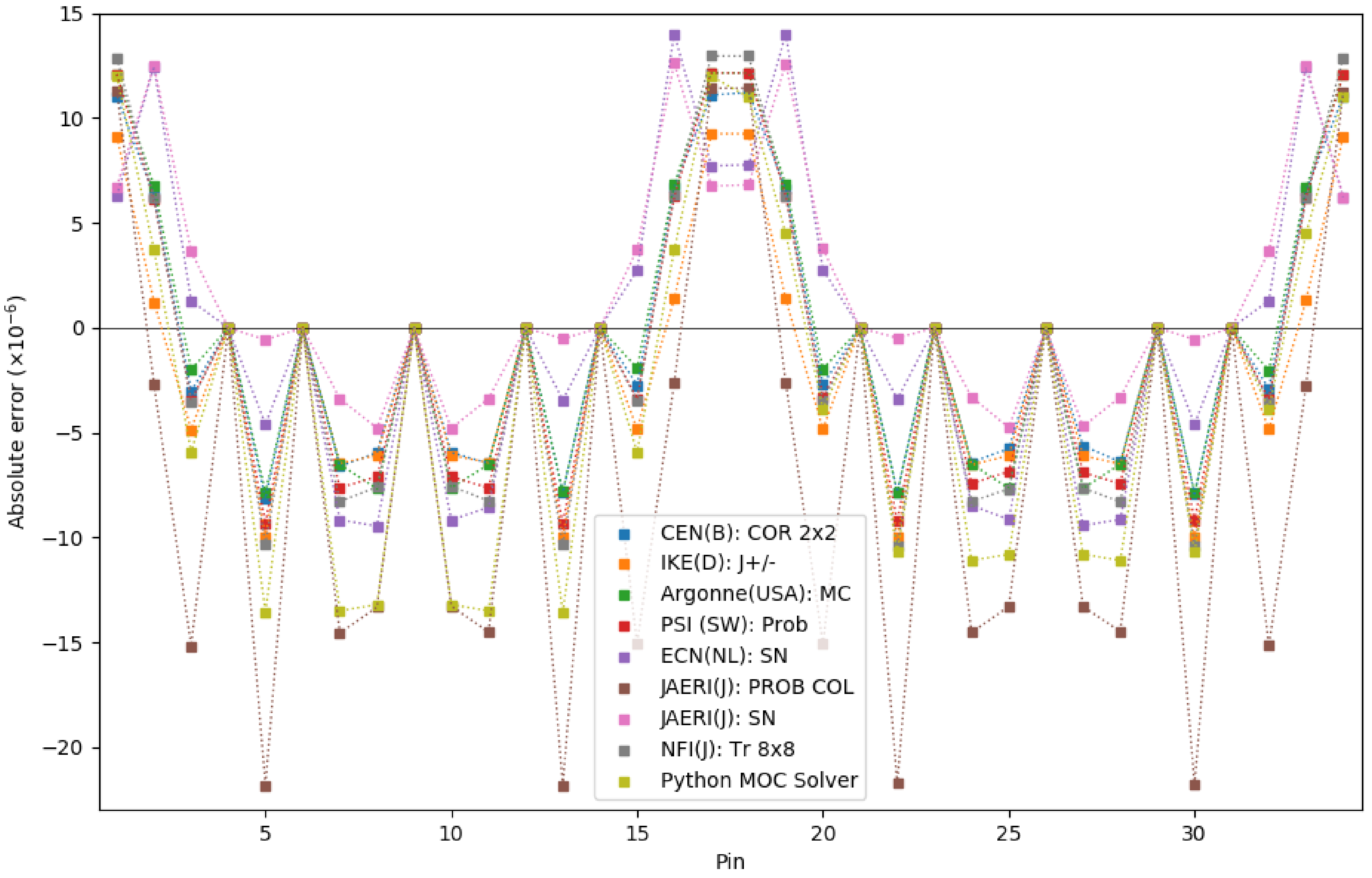
(a) UX Assembly

Key
Calculated Reference Difference

								1208 1224 -1.31%
							1194 1207 -1.08%	1153 1164 -0.95%
						1096 1092 0.37%	1144 1148 -0.35%	1144 1151 -0.61%
					-	1124 1113 0.99%	1140 1146 -0.52%	1147 1156 -0.78%
				1151 1150 0.09%	1091 1076 1.39%	1135 1125 0.89%	1162 1168 -0.51%	1156 1167 -0.94%
			-	1133 1125 0.71%	1182 1178 0.34%	-	1200 1200 0.00%	1164 1174 -0.85%
		1040 1036 0.39%	1097 1084 1.20%	1067 1061 0.57%	1108 1106 0.18%	1092 1077 1.39%	1158 1164 -0.52%	1161 1173 -1.02%
	1035 1031 0.39%	1037 1032 0.48%	1092 1079 1.20%	1060 1053 0.66%	1101 1098 0.27%	1087 1071 1.49%	1156 1162 -0.52%	1162 1173 -0.94%
-	1082 1070 1.12%	1086 1072 1.31%	-	1102 1091 1.01%	1143 1134 0.79%	-	1192 1192 0.00%	1165 1176 -0.94%

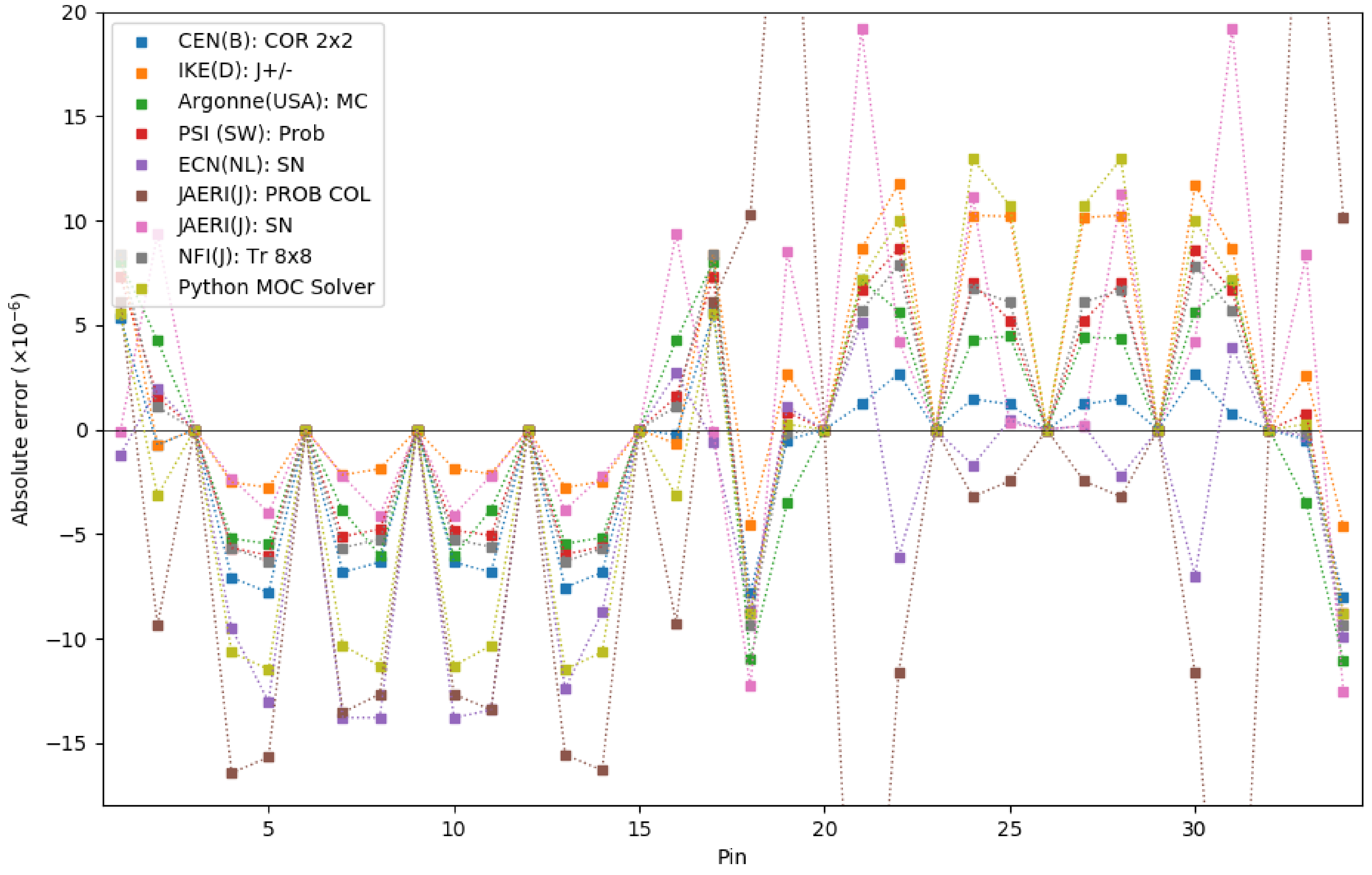
(b) PX Assembly

Figure 4.15: Comparison of Heterogeneous Production Rate Distribution ($\times 10^{-6}$) for C2 Configuration



(a) Absolute Error along TR1

Figure 4.16: Comparison of Pin Power Error for C2 Configuration with other transport calculations



(b) Absolute Error along TR2

Figure 4.16: Comparison of Pin Power Error for C2 Configuration with other transport calculations (cont.)

802 821 -2.31%	805 824 -2.31%	810 829 -2.29%	817 837 -2.39%	825 844 -2.25%	831 850 -2.24%	834 852 -2.11%	837 855 -2.11%	840 858 -2.10%	842 859 -1.98%	844 860 -1.86%	845 859 -1.63%	839 852 -1.53%	825 836 -1.32%	801 809 -0.99%	759 761 -0.26%	677 671 0.89%
805 824 -2.31%	809 828 -2.29%	817 837 -2.39%	827 848 -2.48%	840 862 -2.55%	855 875 -2.29%	848 868 -2.30%	851 870 -2.18%	865 881 -1.82%	857 874 -1.95%	859 876 -1.94%	869 884 -1.70%	854 869 -1.73%	835 848 -1.53%	808 817 -1.10%	762 765 -0.39%	678 672 0.89%
810 829 -2.29%	817 837 -2.39%	833 855 -2.57%	856 877 -2.39%	869 890 -2.36%	-	868 888 -2.25%	870 889 -2.14%	-	876 893 -1.90%	879 896 -1.90%	-	883 898 -1.67%	864 876 -1.37%	823 834 -1.32%	769 772 -0.39%	681 675 0.89%
817 837 -2.39%	827 848 -2.48%	856 877 -2.39%	-	879 902 -2.55%	876 898 -2.45%	864 886 -2.48%	865 886 -2.37%	879 897 -2.01%	871 890 -2.13%	875 893 -2.02%	890 907 -1.87%	892 909 -1.87%	-	846 855 -1.05%	778 782 -0.51%	685 680 0.74%
825 844 -2.25%	840 862 -2.55%	869 890 -2.36%	879 902 -2.55%	874 898 -2.67%	879 900 -2.33%	868 889 -2.36%	869 890 -2.36%	883 901 -2.00%	875 893 -2.02%	878 896 -2.01%	892 909 -1.87%	886 904 -1.99%	885 900 -1.67%	857 867 -1.15%	789 794 -0.63%	691 686 0.73%
831 850 -2.24%	855 875 -2.29%	-	876 898 -2.45%	880 900 -2.22%	-	881 900 -2.11%	883 901 -2.00%	-	888 905 -1.88%	890 907 -1.87%	-	891 906 -1.66%	881 894 -1.45%	-	802 805 -0.37%	694 690 0.58%
834 852 -2.11%	848 868 -2.30%	868 888 -2.25%	864 886 -2.48%	868 889 -2.36%	881 900 -2.11%	873 892 -2.13%	875 894 -2.13%	888 905 -1.88%	880 897 -1.90%	882 899 -1.89%	892 908 -1.76%	878 894 -1.79%	867 882 -1.70%	852 863 -1.27%	794 798 -0.50%	695 690 0.72%
837 855 -2.11%	851 870 -2.18%	870 889 -2.14%	865 886 -2.37%	869 890 -2.36%	883 901 -2.00%	875 894 -2.13%	877 895 -2.01%	890 907 -1.87%	882 899 -1.89%	883 900 -1.89%	894 908 -1.54%	878 894 -1.79%	867 881 -1.59%	853 863 -1.16%	794 798 -0.50%	695 690 0.72%
840 858 -2.10%	865 881 -1.82%	-	879 897 -2.01%	883 901 -2.00%	-	888 905 -1.88%	890 907 -1.87%	-	895 910 -1.65%	896 910 -1.54%	-	891 904 -1.44%	879 891 -1.35%	-	805 807 -0.25%	696 692 0.58%
842 859 -1.98%	857 874 -1.95%	876 893 -1.90%	871 890 -2.13%	875 893 -2.02%	888 905 -1.88%	880 897 -1.90%	882 899 -1.89%	895 910 -1.65%	885 902 -1.88%	886 902 -1.77%	896 911 -1.65%	881 896 -1.67%	869 883 -1.59%	855 864 -1.04%	795 799 -0.50%	696 691 0.72%
844 860 -1.86%	859 876 -1.94%	879 896 -1.90%	875 893 -2.02%	878 896 -2.01%	890 907 -1.87%	882 899 -1.89%	883 900 -1.89%	896 910 -1.54%	886 902 -1.77%	888 903 -1.66%	897 912 -1.64%	883 898 -1.67%	871 885 -1.58%	856 865 -1.04%	796 800 -0.50%	696 692 0.58%
844 860 -1.86%	869 884 -1.70%	-	890 907 -1.87%	892 909 -1.87%	-	892 908 -1.76%	894 908 -1.54%	-	896 911 -1.65%	897 912 -1.64%	-	896 910 -1.54%	884 897 -1.45%	-	805 808 -0.37%	696 692 0.58%
839 852 -1.53%	854 869 -1.73%	883 898 -1.67%	892 909 -1.87%	886 904 -1.99%	891 906 -1.66%	878 894 -1.79%	878 894 -1.79%	891 904 -1.44%	881 896 -1.67%	883 898 -1.67%	896 910 -1.54%	888 904 -1.77%	886 899 -1.45%	858 868 -1.15%	790 795 -0.63%	692 687 0.73%
825 837 -1.43%	835 848 -1.53%	864 876 -1.37%	-	885 900 -1.67%	881 895 -1.56%	867 882 -1.70%	867 881 -1.59%	879 891 -1.35%	869 883 -1.59%	871 885 -1.58%	884 897 -1.45%	886 899 -1.45%	-	840 848 -0.94%	775 778 -0.39%	684 679 0.74%
801 810 -1.11%	808 817 -1.10%	823 834 -1.32%	846 855 -1.05%	857 868 -1.27%	-	852 863 -1.27%	853 863 -1.16%	-	855 864 -1.04%	856 865 -1.04%	-	858 868 -1.15%	840 848 -0.94%	804 810 -0.74%	755 756 -0.13%	676 669 1.05%
759 761 -0.26%	762 765 -0.39%	769 772 -0.39%	778 782 -0.51%	789 794 -0.63%	802 805 -0.37%	793 798 -0.63%	794 798 -0.50%	805 807 -0.25%	795 800 -0.63%	796 800 -0.50%	805 808 -0.37%	790 795 -0.63%	775 778 -0.39%	755 756 -0.13%	725 721 0.55%	662 653 1.38%
677 671 0.89%	678 672 0.89%	681 675 0.89%	685 680 0.74%	691 686 0.73%	694 690 0.58%	694 690 0.58%	695 690 0.72%	696 692 0.58%	696 691 0.72%	696 692 0.58%	696 692 0.58%	692 687 0.73%	684 679 0.74%	676 669 1.05%	662 653 1.38%	634 624 1.60%

Key
Calculated
Reference
Difference

(a) UX Assembly

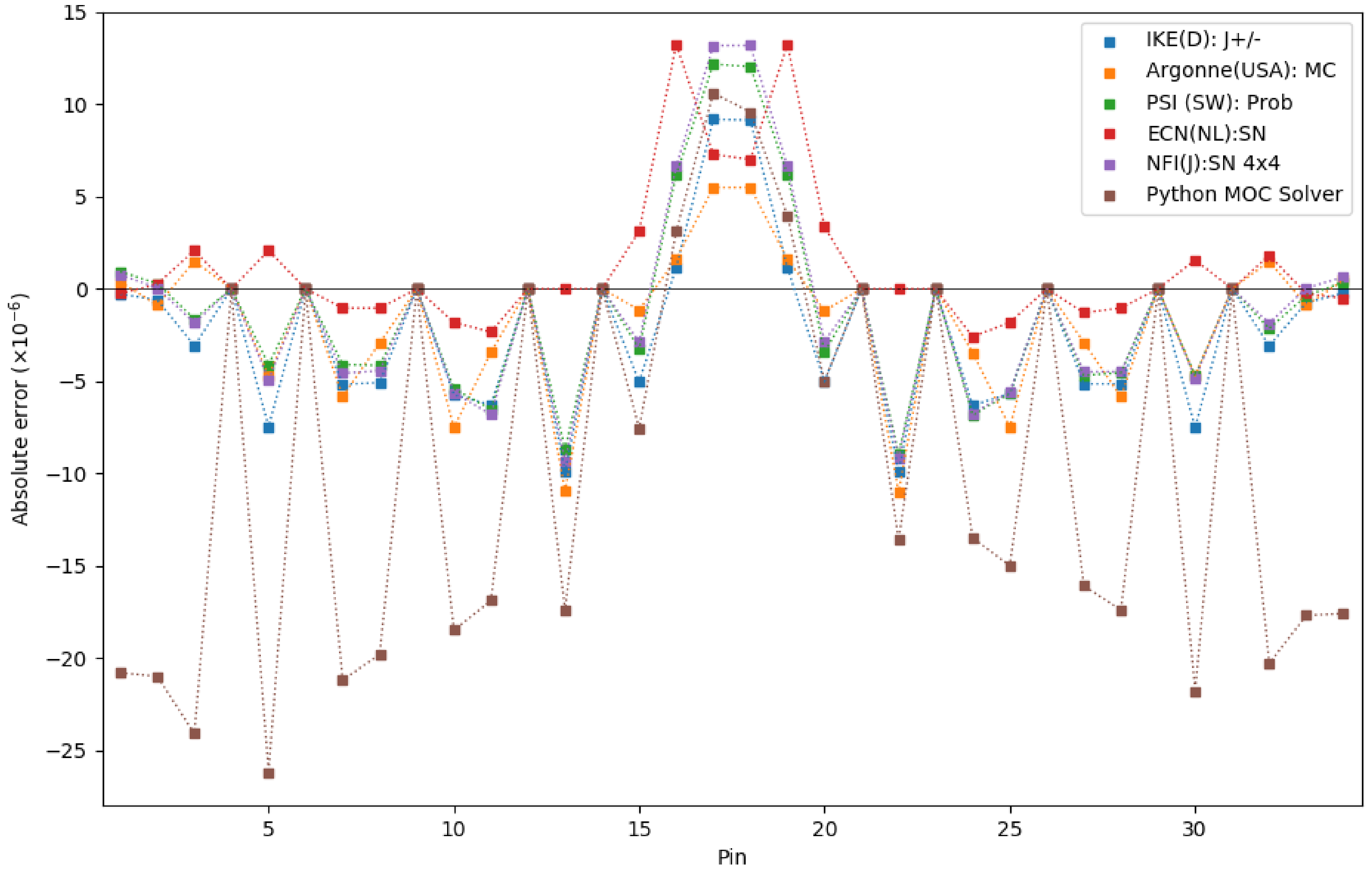
Figure 4.17: Comparison of Heterogeneous Production Rate Distribution ($\times 10^{-6}$) for C3 Configuration

1153	989	927	907	906	909	908	909	913	913	915	919	915	909	906	909	917
1164	982	911	890	889	892	890	890	894	892	894	897	892	885	881	885	894
-0.95%	0.71%	1.76%	1.91%	1.91%	1.91%	2.02%	2.13%	2.13%	2.35%	2.35%	2.45%	2.58%	2.71%	2.84%	2.71%	2.57%
1147	1135	1057	1040	1054	1090	1051	1051	1092	1056	1061	1101	1065	1042	1030	1036	909
1156	1139	1049	1032	1048	1076	1041	1040	1071	1042	1046	1081	1050	1025	1013	1019	885
-0.78%	-0.35%	0.76%	0.78%	0.57%	1.30%	0.96%	1.06%	1.96%	1.34%	1.43%	1.85%	1.43%	1.66%	1.68%	1.67%	2.71%
1146	1129	1071	1095	1104		1058	1057		1061	1066		1114	1098	1045	1030	906
1155	1133	1064	1081	1089	-	1042	1038	-	1040	1046	-	1092	1074	1028	1013	881
-0.78%	-0.35%	0.66%	1.30%	1.38%		1.54%	1.83%		2.02%	1.91%		2.01%	2.23%	1.65%	1.68%	2.84%
1153	1143	1125		1088	1182	1113	1105	1154	1110	1121	1196	1099		1099	1042	909
1163	1146	1111	-	1072	1173	1103	1096	1134	1098	1108	1179	1075	-	1074	1025	885
-0.86%	-0.26%	1.26%		1.49%	0.77%	0.91%	0.82%	1.76%	1.09%	1.17%	1.44%	2.23%		2.33%	1.66%	2.71%
1162	1169	1145	1098	1161	1150	1086	1079	1129	1083	1093	1162	1171	1099	1114	1065	915
1173	1174	1131	1083	1158	1136	1072	1066	1107	1068	1076	1141	1161	1075	1092	1050	892
-0.94%	-0.43%	1.24%	1.39%	0.26%	1.23%	1.31%	1.22%	1.99%	1.40%	1.58%	1.84%	0.86%	2.23%	2.01%	1.43%	2.58%
1169	1209		1200	1153		1116	1114		1120	1125		1162	1196		1101	919
1181	1208	-	1190	1139	-	1100	1097	-	1099	1104	-	1141	1179	-	1081	897
-1.02%	0.08%		0.84%	1.23%		1.45%	1.55%		1.91%	1.90%		1.84%	1.44%		1.85%	2.45%
1169	1170	1100	1126	1085	1116	1068	1066	1111	1070	1075	1125	1093	1121	1066	1061	915
1180	1173	1087	1119	1075	1100	1053	1051	1093	1053	1057	1104	1076	1108	1046	1046	894
-0.93%	-0.26%	1.20%	0.63%	0.93%	1.45%	1.42%	1.43%	1.65%	1.61%	1.70%	1.90%	1.58%	1.17%	1.91%	1.43%	2.35%
1169	1167	1095	1118	1078	1112	1063	1062	1109	1066	1070	1120	1083	1110	1061	1056	913
1180	1171	1082	1111	1068	1095	1050	1049	1091	1051	1053	1099	1068	1098	1040	1042	892
-0.93%	-0.34%	1.20%	0.63%	0.94%	1.55%	1.24%	1.24%	1.65%	1.43%	1.61%	1.91%	1.40%	1.09%	2.02%	1.34%	2.35%
1172	1206		1164	1126		1106	1108		1109	1111		1129	1154		1092	913
1183	1203	-	1149	1107	-	1090	1089	-	1091	1093	-	1107	1134	-	1071	894
-0.93%	0.25%		1.31%	1.72%		1.47%	1.74%		1.65%	1.65%		1.99%	1.76%		1.96%	2.13%
1170	1166	1096	1117	1077	1109	1061	1059	1107	1062	1067	1114	1079	1105	1057	1051	909
1181	1172	1082	1111	1067	1095	1049	1048	1089	1049	1051	1097	1066	1096	1038	1040	890
-0.93%	-0.51%	1.29%	0.54%	0.94%	1.28%	1.14%	1.05%	1.65%	1.24%	1.52%	1.55%	1.22%	0.82%	1.83%	1.06%	2.13%
1171	1170	1101	1125	1085	1112	1065	1061	1106	1063	1068	1116	1085	1113	1058	1051	908
1182	1174	1088	1120	1075	1099	1051	1049	1090	1050	1053	1100	1072	1103	1042	1041	890
-0.93%	-0.34%	1.19%	0.45%	0.93%	1.18%	1.33%	1.14%	1.47%	1.24%	1.42%	1.45%	1.21%	0.91%	1.54%	0.96%	2.02%
1173	1212		1199	1151		1112	1109		1112	1116		1150	1182		1090	909
1184	1211	-	1192	1140	-	1099	1095	-	1095	1100	-	1136	1173	-	1076	892
-0.93%	0.08%		0.59%	0.96%		1.18%	1.28%		1.55%	1.45%		1.23%	0.77%		1.30%	1.91%
1166	1173	1150	1102	1164	1151	1085	1077	1126	1078	1085	1153	1161	1088	1104	1054	906
1177	1180	1137	1089	1164	1140	1075	1067	1107	1068	1075	1139	1158	1072	1089	1048	889
-0.93%	-0.59%	1.14%	1.19%	0.00%	0.96%	0.93%	0.94%	1.72%	0.94%	0.93%	1.23%	0.26%	1.49%	1.38%	0.57%	1.91%
1158	1153	1137		1102	1199	1125	1117	1164	1118	1126	1199	1098		1095	1040	907
1167	1157	1126	-	1089	1192	1120	1111	1149	1111	1119	1190	1083	-	1081	1032	890
-0.77%	-0.35%	0.98%		1.19%	0.59%	0.45%	0.54%	1.31%	0.63%	0.63%	0.76%	1.39%		1.30%	0.78%	1.91%
1153	1155	1109	1137	1150		1101	1096		1095	1100		1145	1125	1071	1057	927
1162	1160	1103	1126	1137	-	1088	1082	-	1082	1087	-	1131	1111	1064	1049	911
-0.77%	-0.43%	0.54%	0.98%	1.14%		1.19%	1.29%		1.20%	1.20%		1.24%	1.26%	0.66%	0.76%	1.76%
1165	1206	1155	1153	1173	1212	1170	1167	1206	1167	1170	1209	1168	1143	1129	1135	989
1175	1218	1160	1157	1180	1211	1174	1172	1203	1171	1173	1208	1174	1146	1133	1139	982
-0.85%	-0.99%	-0.43%	-0.35%	-0.59%	0.08%	-0.34%	-0.43%	0.25%	-0.34%	-0.26%	0.08%	-0.51%	-0.26%	-0.35%	-0.35%	0.71%
1221	1165	1153	1158	1165	1173	1171	1170	1172	1169	1169	1169	1162	1153	1146	1147	1153
1235	1175	1162	1167	1177	1184	1182	1181	1183	1180	1180	1181	1173	1163	1155	1156	1164
-1.13%	-0.85%	-0.77%	-0.77%	-1.02%	-0.93%	-0.93%	-0.93%	-0.93%	-0.93%	-0.93%	-1.02%	-0.94%	-0.86%	-0.78%	-0.78%	-0.95%

Key
Calculated
Reference
Difference

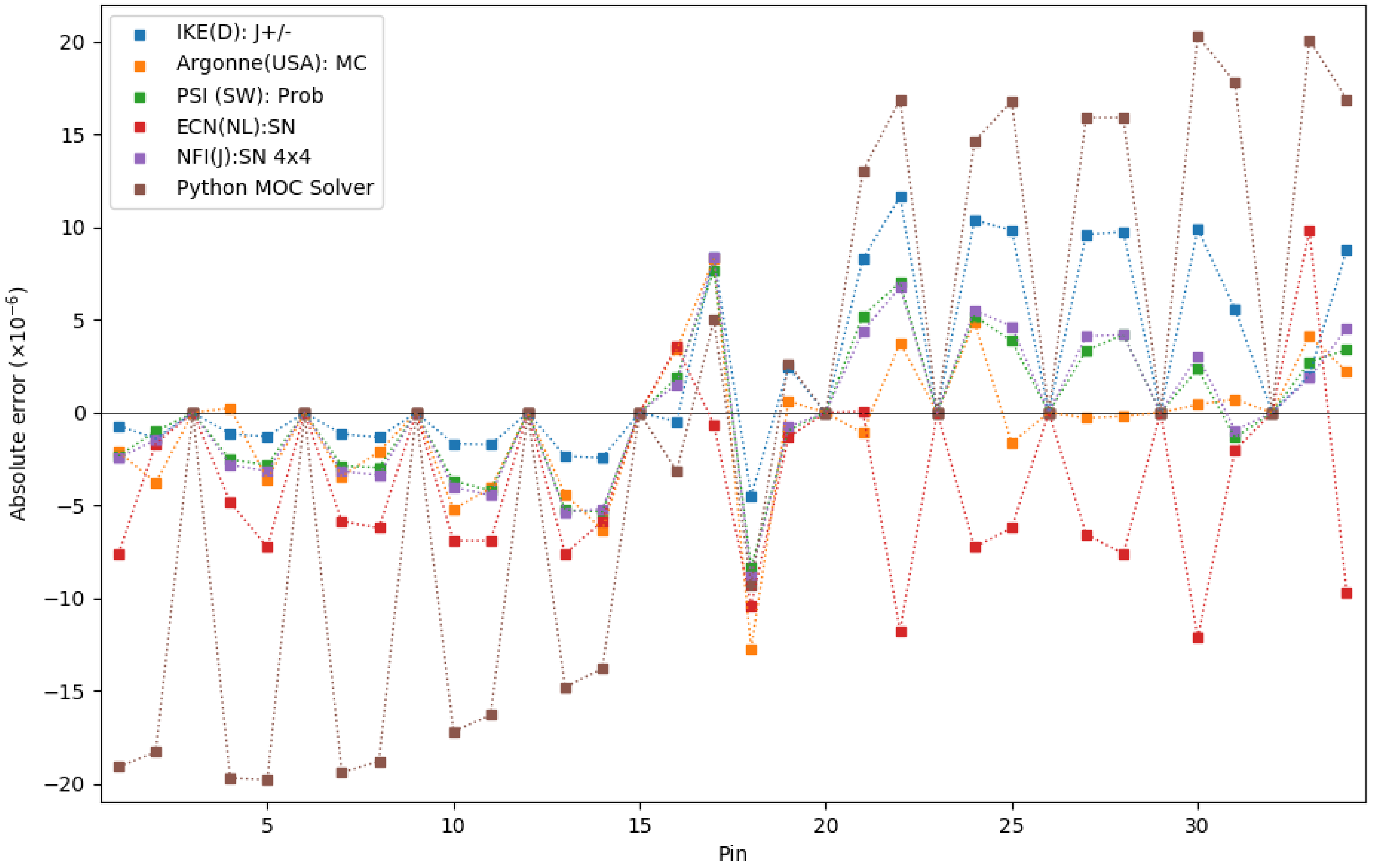
(b) PX Assembly

Figure 4.17: Comparison of Heterogeneous Production Rate Distribution ($\times 10^{-6}$) for C3 Configuration (cont.)



(a) Absolute Error along TR1

Figure 4.18: Comparison of Pin Power Error for C3 Configuration with other transport calculations



(b) Absolute Error along TR2

Figure 4.18: Comparison of Pin Power Error for C3 Configuration with other transport calculations (cont.)

1640	1642	1645	1648	1649	1643	1628	1610	1590	1563	1534	1499	1453	1392	1316	1210	1047
1625	1628	1632	1637	1639	1635	1620	1602	1582	1556	1528	1495	1448	1387	1308	1197	1025
0.92%	0.86%	0.80%	0.67%	0.61%	0.49%	0.49%	0.50%	0.51%	0.45%	0.39%	0.27%	0.35%	0.36%	0.61%	1.09%	2.15%
1642	1646	1654	1664	1676	1687	1654	1633	1631	1586	1557	1540	1476	1406	1323	1213	1046
1628	1633	1643	1655	1669	1677	1646	1627	1621	1581	1553	1534	1475	1403	1317	1200	1024
0.86%	0.80%	0.67%	0.54%	0.42%	0.60%	0.49%	0.37%	0.62%	0.32%	0.26%	0.39%	0.07%	0.21%	0.46%	1.08%	2.15%
1645	1654	1679	1715	1725		1684	1663		1615	1585		1520	1448	1344	1219	1048
1632	1643	1671	1704	1717	-	1677	1656	-	1609	1581	-	1517	1445	1340	1208	1025
0.80%	0.67%	0.48%	0.65%	0.47%		0.42%	0.42%		0.37%	0.25%		0.20%	0.21%	0.30%	0.91%	2.24%
1648	1664	1715		1734	1711	1666	1644	1642	1596	1569	1559	1526		1372	1227	1050
1637	1655	1704	-	1729	1705	1663	1641	1635	1594	1567	1558	1526	-	1367	1217	1027
0.67%	0.54%	0.65%		0.29%	0.35%	0.18%	0.18%	0.43%	0.13%	0.13%	0.06%	0.00%		0.37%	0.82%	2.24%
1649	1676	1725	1734	1710	1702	1659	1637	1635	1589	1562	1551	1504	1462	1379	1235	1050
1639	1669	1717	1729	1708	1697	1656	1635	1629	1588	1561	1550	1507	1463	1376	1226	1029
0.61%	0.42%	0.47%	0.29%	0.12%	0.29%	0.18%	0.12%	0.37%	0.06%	0.06%	0.06%	-0.20%	-0.07%	0.22%	0.73%	2.04%
1643	1687		1711	1702		1666	1645		1597	1567		1498	1441		1243	1046
1635	1677	-	1705	1697	-	1661	1640	-	1594	1565	-	1497	1441	-	1233	1025
0.49%	0.60%		0.35%	0.29%		0.30%	0.30%		0.19%	0.13%		0.07%	0.00%		0.81%	2.05%
1628	1654	1684	1666	1659	1666	1631	1611	1607	1564	1535	1518	1459	1403	1343	1216	1034
1620	1646	1677	1663	1656	1661	1627	1608	1603	1562	1534	1517	1461	1405	1341	1208	1014
0.49%	0.49%	0.42%	0.18%	0.18%	0.30%	0.25%	0.19%	0.25%	0.13%	0.07%	0.07%	-0.14%	-0.14%	0.15%	0.66%	1.97%
1610	1633	1663	1644	1637	1645	1611	1591	1588	1545	1515	1499	1439	1384	1325	1200	1021
1602	1627	1656	1641	1635	1640	1608	1589	1584	1544	1515	1497	1441	1386	1323	1192	1002
0.50%	0.37%	0.42%	0.18%	0.12%	0.30%	0.19%	0.13%	0.25%	0.06%	0.00%	0.13%	-0.14%	-0.14%	0.15%	0.67%	1.90%
1590	1631		1642	1635		1607	1588		1543	1512		1437	1381		1199	1007
1582	1621	-	1635	1629	-	1603	1584	-	1539	1510	-	1436	1380	-	1188	989
0.51%	0.62%		0.43%	0.37%		0.25%	0.25%		0.26%	0.13%		0.07%	0.07%		0.93%	1.82%
1563	1586	1615	1596	1589	1597	1564	1545	1543	1498	1470	1454	1396	1343	1286	1164	990
1556	1581	1609	1594	1588	1594	1562	1544	1539	1499	1471	1454	1399	1345	1284	1157	972
0.45%	0.32%	0.37%	0.13%	0.06%	0.19%	0.13%	0.06%	0.26%	-0.07%	-0.07%	0.00%	-0.21%	-0.15%	0.16%	0.61%	1.85%
1534	1557	1585	1569	1562	1567	1535	1515	1512	1470	1443	1426	1371	1319	1261	1142	971
1528	1553	1581	1567	1561	1565	1534	1515	1510	1471	1444	1427	1374	1322	1261	1135	954
0.39%	0.26%	0.25%	0.13%	0.06%	0.13%	0.07%	0.00%	0.13%	-0.07%	-0.07%	-0.07%	-0.22%	-0.23%	0.00%	0.62%	1.78%
1499	1540		1559	1551		1518	1499		1454	1426		1361	1310		1131	951
1495	1534	-	1558	1550	-	1517	1497	-	1454	1427	-	1364	1313	-	1123	934
0.27%	0.39%		0.06%	0.06%		0.07%	0.13%		0.00%	-0.07%		-0.22%	-0.23%		0.71%	1.82%
1453	1476	1520	1526	1504	1498	1459	1439	1437	1396	1371	1361	1319	1282	1210	1085	924
1448	1475	1517	1526	1507	1497	1461	1441	1436	1399	1374	1364	1325	1286	1211	1080	907
0.35%	0.07%	0.20%	0.00%	-0.20%	0.07%	-0.14%	-0.14%	0.07%	-0.21%	-0.22%	-0.22%	-0.45%	-0.31%	-0.08%	0.46%	1.87%
1392	1406	1448		1462	1441	1403	1384	1381	1343	1319	1310	1282		1155	1036	890
1387	1403	1445	-	1463	1441	1405	1386	1380	1345	1322	1313	1286	-	1155	1031	873
0.36%	0.21%	0.21%		-0.07%	0.00%	-0.14%	-0.14%	0.07%	-0.15%	-0.23%	-0.23%	-0.31%		0.00%	0.48%	1.95%
1316	1323	1344	1372	1379		1343	1325		1286	1261		1210	1155	1076	982	852
1308	1317	1340	1367	1376	-	1341	1323	-	1284	1261	-	1211	1155	1075	974	835
0.61%	0.46%	0.30%	0.37%	0.22%		0.15%	0.15%		0.16%	0.00%		-0.08%	0.00%	0.09%	0.82%	2.04%
1210	1213	1219	1227	1235	1243	1216	1200	1199	1164	1142	1131	1085	1036	982	913	807
1197	1200	1208	1217	1226	1233	1208	1192	1188	1157	1135	1123	1080	1031	974	901	789
1.09%	1.08%	0.91%	0.82%	0.73%	0.81%	0.66%	0.67%	0.93%	0.61%	0.62%	0.71%	0.46%	0.48%	0.82%	1.33%	2.28%
1047	1046	1048	1050	1050	1046	1034	1021	1007	990	971	951	924	890	852	807	744
1025	1024	1025	1027	1029	1025	1014	1002	989	972	954	934	907	873	835	789	724
2.15%	2.15%	2.24%	2.24%	2.04%	2.05%	1.97%	1.90%	1.82%	1.85%	1.78%	1.82%	1.87%	1.95%	2.04%	2.28%	2.76%

Key
Calculated
Reference
Difference

(a) Top Left UX Assembly

Figure 4.19: Comparison of Heterogeneous Production Rate Distribution ($\times 10^{-6}$) for C4V Configuration

1720 1717 0.17%	1408 1384 1.73%	1252 1220 2.62%	1157 1128 2.57%	1089 1064 2.35%	1024 1005 1.89%	955 937 1.92%	885 869 1.84%	816 803 1.62%	741 731 1.37%	667 658 1.37%	592 586 1.02%	512 508 0.79%	429 427 0.47%	343 344 -0.29%	254 258 -1.55%	153 163 -6.13%
1709 1705 0.23%	1613 1605 0.50%	1424 1403 1.50%	1324 1306 1.38%	1263 1251 0.96%	1228 1210 1.49%	1105 1096 0.82%	1020 1013 0.69%	974 961 1.35%	857 853 0.47%	770 767 0.39%	708 705 0.43%	596 599 -0.50%	491 494 -0.61%	389 395 -1.52%	288 296 -2.70%	151 160 -5.63%
1704 1698 0.35%	1600 1590 0.63%	1437 1416 1.48%	1390 1362 2.06%	1319 1297 1.70%	-	1112 1095 1.55%	1021 1006 1.49%	-	861 850 1.29%	770 763 0.92%	-	622 619 0.48%	515 516 -0.19%	396 402 -1.49%	286 295 -3.05%	150 159 -5.66%
1705 1699 0.35%	1609 1600 0.56%	1497 1468 1.98%	-	1301 1276 1.96%	1326 1313 0.99%	1163 1154 0.78%	1064 1059 0.47%	1021 1010 1.09%	894 892 0.22%	806 805 0.12%	759 757 0.26%	602 600 0.33%	-	417 420 -0.71%	290 298 -2.68%	150 160 -6.25%
1707 1702 0.29%	1633 1626 0.43%	1515 1488 1.81%	1381 1355 1.92%	1378 1370 0.58%	1280 1261 1.51%	1128 1114 1.26%	1033 1023 0.98%	993 978 1.53%	867 862 0.58%	780 776 0.52%	732 728 0.55%	639 644 -0.78%	503 504 -0.20%	416 420 -0.95%	296 304 -2.63%	152 161 -5.59%
1700 1697 0.18%	1672 1657 0.91%	-	1501 1479 1.49%	1352 1331 1.58%	-	1150 1134 1.41%	1056 1041 1.44%	-	892 881 1.25%	795 788 0.89%	-	634 632 0.32%	539 543 -0.74%	-	305 311 -1.93%	151 160 -5.63%
1681 1678 0.18%	1605 1595 0.63%	1435 1408 1.92%	1393 1377 1.16%	1262 1245 1.37%	1218 1197 1.75%	1088 1073 1.40%	1002 990 1.21%	961 948 1.37%	842 835 0.84%	755 749 0.80%	700 695 0.72%	589 589 0.00%	502 506 -0.79%	386 389 -0.77%	287 295 -2.71%	148 157 -5.73%
1657 1657 0.00%	1580 1572 0.51%	1412 1383 2.10%	1367 1348 1.41%	1237 1220 1.39%	1198 1176 1.87%	1068 1055 1.23%	986 975 1.13%	947 933 1.50%	829 822 0.85%	742 737 0.68%	688 682 0.88%	577 577 0.00%	491 495 -0.81%	378 380 -0.53%	280 289 -3.11%	146 155 -5.81%
1637 1637 0.00%	1607 1590 1.07%	-	1403 1376 1.96%	1270 1244 2.09%	-	1099 1081 1.67%	1012 996 1.61%	-	853 842 1.31%	759 752 0.93%	-	597 592 0.84%	501 502 -0.20%	-	289 295 -2.03%	143 153 -6.54%
1607 1607 0.00%	1532 1525 0.46%	1370 1342 2.09%	1324 1308 1.22%	1199 1182 1.44%	1160 1140 1.75%	1036 1023 1.27%	955 945 1.06%	918 905 1.44%	802 796 0.75%	718 714 0.56%	666 661 0.76%	559 559 0.00%	475 479 -0.84%	366 368 -0.54%	271 280 -3.21%	141 150 -6.00%
1578 1577 0.06%	1505 1499 0.40%	1345 1321 1.82%	1306 1292 1.08%	1184 1167 1.46%	1140 1122 1.60%	1020 1005 1.49%	938 927 1.19%	900 887 1.47%	787 781 0.77%	706 701 0.71%	653 649 0.62%	551 551 0.00%	469 473 -0.85%	360 363 -0.83%	268 275 -2.55%	138 147 -6.12%
1549 1547 0.13%	1525 1510 0.99%	-	1363 1346 1.26%	1228 1210 1.49%	-	1046 1031 1.45%	959 946 1.37%	-	810 799 1.38%	721 715 0.84%	-	573 572 0.17%	487 491 -0.81%	-	276 282 -2.13%	136 145 -6.21%
1502 1502 0.00%	1442 1438 0.28%	1341 1317 1.82%	1220 1197 1.92%	1216 1210 0.50%	1131 1115 1.43%	994 983 1.12%	911 903 0.89%	876 863 1.51%	764 759 0.66%	687 683 0.59%	645 641 0.62%	561 565 -0.71%	441 442 -0.23%	365 368 -0.82%	260 267 -2.62%	133 141 -5.67%
1453 1450 0.21%	1380 1373 0.51%	1287 1265 1.74%	-	1120 1100 1.82%	1144 1131 1.15%	1001 994 0.70%	918 911 0.77%	878 867 1.27%	769 766 0.39%	692 690 0.29%	651 648 0.46%	515 513 0.39%	-	356 359 -0.84%	248 254 -2.36%	128 136 -5.88%
1401 1400 0.07%	1337 1333 0.30%	1216 1202 1.16%	1183 1163 1.72%	1127 1109 1.62%	-	949 936 1.39%	870 858 1.40%	-	732 723 1.24%	654 648 0.93%	-	527 525 0.38%	437 437 0.00%	336 340 -1.18%	242 248 -2.42%	126 133 -5.26%
1363 1364 -0.07%	1342 1345 -0.22%	1218 1214 0.33%	1151 1147 0.35%	1105 1105 0.00%	1076 1069 0.65%	970 970 0.00%	895 896 -0.11%	852 848 0.47%	750 752 -0.27%	672 676 -0.59%	617 619 -0.32%	519 525 -1.14%	427 432 -1.16%	338 345 -2.03%	248 256 -3.13%	128 137 -6.57%
1365 1371 -0.44%	1238 1238 0.00%	1162 1162 0.00%	1105 1106 -0.09%	1051 1055 -0.38%	994 999 -0.50%	928 933 -0.54%	861 865 -0.46%	793 798 -0.63%	719 725 -0.83%	645 652 -1.07%	572 579 -1.21%	493 500 -1.40%	411 418 -1.67%	327 335 -2.39%	240 248 -3.23%	142 152 -6.58%

Key

Calculated
Reference
Difference

(b) Top Right PX Assembly

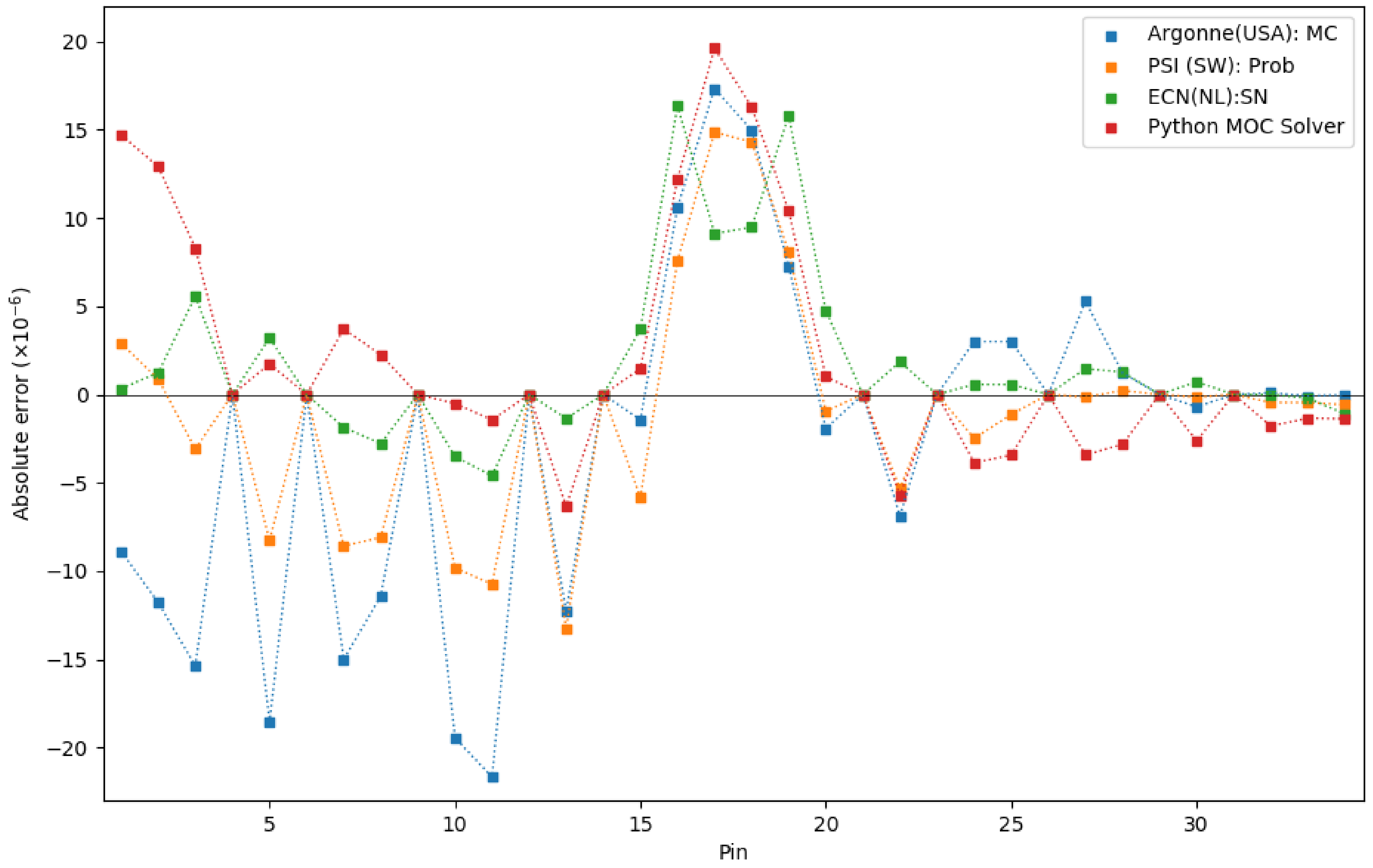
Figure 4.19: Comparison of Heterogeneous Production Rate Distribution ($\times 10^{-6}$) for C4V Configuration (cont.)

676 660 2.42%	673 659 2.12%	653 641 1.87%	627 618 1.46%	600 591 1.52%	567 561 1.07%	531 524 1.34%	493 487 1.23%	454 448 1.34%	412 408 0.98%	369 367 0.54%	327 325 0.62%	282 281 0.36%	235 235 0.00%	187 188 -0.53%	136 139 -2.16%	80 85 -5.88%
673 659 2.12%	704 694 1.44%	700 694 0.86%	682 679 0.44%	658 656 0.30%	631 628 0.48%	584 583 0.17%	541 540 0.19%	504 502 0.40%	453 453 0.00%	406 407 -0.25%	363 364 -0.27%	310 312 -0.64%	257 259 -0.77%	203 206 -1.46%	147 151 -2.65%	85 91 -6.59%
653 641 1.87%	700 694 0.86%	711 710 0.14%	707 707 0.00%	685 686 -0.15%	-	603 605 -0.33%	558 560 -0.36%	-	467 469 -0.43%	418 421 -0.71%	-	323 326 -0.92%	268 271 -1.11%	209 214 -2.34%	150 154 -2.60%	86 92 -6.52%
627 618 1.46%	682 679 0.44%	707 707 0.00%	-	676 679 -0.44%	636 639 -0.47%	585 590 -0.85%	541 546 -0.92%	504 507 -0.59%	453 457 -0.88%	406 410 -0.98%	365 369 -1.08%	317 322 -1.55%	-	210 214 -1.87%	148 153 -3.27%	84 90 -6.67%
600 591 1.52%	658 656 0.30%	685 686 -0.15%	676 679 -0.44%	642 648 -0.93%	610 614 -0.65%	562 567 -0.88%	520 525 -0.95%	484 487 -0.62%	435 439 -0.91%	390 394 -1.02%	350 354 -1.13%	302 307 -1.63%	255 260 -1.92%	203 207 -1.93%	144 149 -3.36%	82 87 -5.75%
567 561 1.07%	631 628 0.48%	-	636 639 -0.47%	610 614 -0.65%	-	539 543 -0.74%	499 503 -0.80%	-	418 421 -0.71%	374 377 -0.80%	-	287 291 -1.37%	239 244 -2.05%	-	139 142 -2.11%	77 83 -7.23%
531 524 1.34%	584 583 0.17%	603 605 -0.33%	585 590 -0.85%	562 567 -0.88%	539 543 -0.74%	499 503 -0.80%	462 466 -0.86%	431 434 -0.69%	387 390 -0.77%	346 350 -1.14%	309 313 -1.28%	264 268 -1.49%	220 224 -1.79%	177 181 -2.21%	127 131 -3.05%	72 77 -6.49%
493 487 1.23%	541 540 0.19%	558 560 -0.36%	541 546 -0.92%	520 525 -0.95%	499 503 -0.80%	462 466 -0.86%	429 432 -0.69%	399 401 -0.50%	358 362 -1.10%	320 324 -1.23%	286 289 -1.04%	244 248 -1.61%	204 207 -1.45%	164 167 -1.80%	117 121 -3.31%	67 71 -5.63%
454 448 1.34%	504 502 0.40%	-	504 507 -0.59%	484 487 -0.62%	-	431 434 -0.69%	399 401 -0.50%	-	333 336 -0.89%	297 301 -1.33%	-	228 230 -0.87%	189 192 -1.56%	-	109 113 -3.54%	61 66 -7.58%
412 408 0.98%	453 453 0.00%	467 469 -0.43%	453 457 -0.88%	435 439 -0.91%	418 421 -0.71%	387 390 -0.77%	358 362 -1.10%	333 336 -0.89%	299 302 -0.99%	267 271 -1.48%	239 242 -1.24%	204 207 -1.45%	170 173 -1.73%	137 140 -2.14%	98 101 -2.97%	56 59 -5.08%
369 367 0.54%	406 407 -0.25%	418 421 -0.71%	406 410 -0.98%	390 394 -1.02%	374 377 -0.80%	346 350 -1.14%	320 324 -1.23%	297 301 -1.33%	267 271 -1.48%	239 242 -1.24%	214 216 -0.93%	183 185 -1.08%	152 155 -1.94%	122 125 -2.40%	88 91 -3.30%	50 53 -5.66%
327 325 0.62%	363 364 -0.27%	-	365 369 -1.08%	350 354 -1.13%	-	309 313 -1.28%	286 289 -1.04%	-	239 242 -1.24%	214 216 -0.93%	-	164 166 -1.20%	136 139 -2.16%	-	79 81 -2.47%	44 47 -6.38%
282 281 0.36%	310 312 -0.64%	323 326 -0.92%	317 322 -1.55%	302 307 -1.63%	287 291 -1.37%	264 268 -1.49%	244 248 -1.61%	228 230 -0.87%	204 207 -1.45%	183 185 -1.08%	164 166 -1.20%	140 143 -2.10%	118 121 -2.48%	94 97 -3.09%	67 70 -4.29%	38 41 -7.32%
235 235 0.00%	257 259 -0.77%	268 271 -1.11%	-	255 260 -1.92%	239 244 -2.05%	220 224 -1.79%	204 207 -1.45%	189 192 -1.56%	170 173 -1.73%	152 155 -1.94%	136 139 -2.16%	118 121 -2.48%	-	79 81 -2.47%	55 57 -3.51%	32 34 -5.88%
187 188 -0.53%	203 206 -1.46%	209 214 -2.34%	210 214 -1.87%	203 207 -1.93%	-	177 181 -2.21%	164 167 -1.80%	-	137 140 -2.14%	122 125 -2.40%	-	94 97 -3.09%	79 81 -2.47%	61 63 -3.17%	44 45 -2.22%	25 27 -7.41%
136 139 -2.16%	147 151 -2.65%	150 154 -2.60%	148 153 -3.27%	144 149 -3.36%	139 142 -2.11%	127 131 -3.05%	117 121 -3.31%	109 113 -3.54%	98 101 -2.97%	88 91 -3.30%	79 81 -2.47%	67 70 -4.29%	55 57 -3.51%	44 45 -2.22%	32 33 -3.03%	18 20 -10.0%
80 85 -5.88%	85 91 -6.59%	86 92 -6.52%	84 90 -6.67%	82 87 -5.75%	77 83 -7.23%	72 77 -6.49%	67 71 -5.63%	61 66 -7.58%	56 59 -5.08%	50 53 -5.66%	44 47 -6.38%	38 41 -7.32%	32 34 -5.88%	25 27 -7.41%	18 20 -10.0%	11 12 -8.33%

Key
Calculated
Reference
Difference

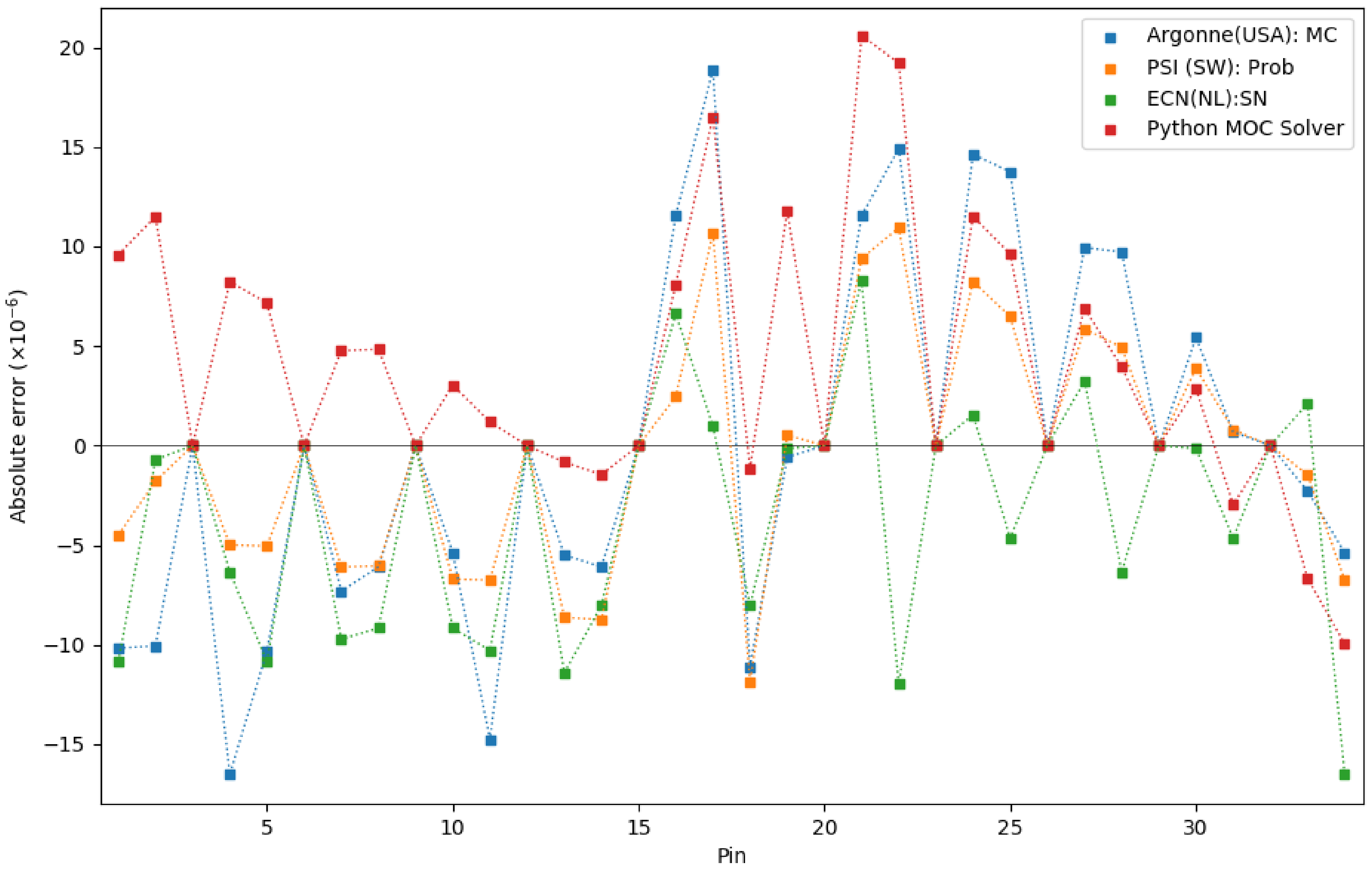
(c) Bottom Right UX Assembly

Figure 4.19: Comparison of Heterogeneous Production Rate Distribution ($\times 10^{-6}$) for C4V Configuration (cont.)



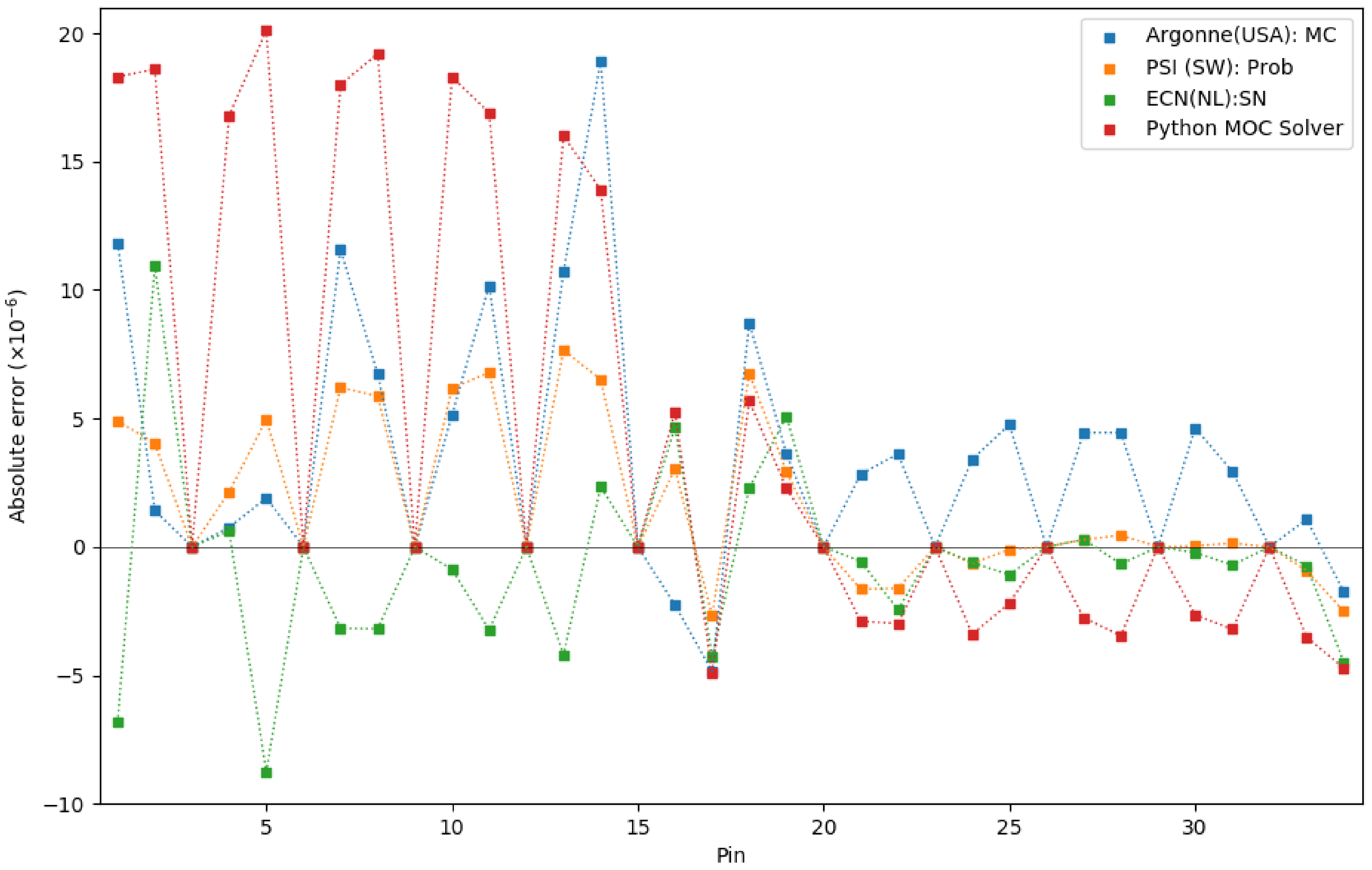
(a) Absolute Error along TR1

Figure 4.20: Comparison of Pin Power Error for C4V Configuration with other transport calculations



(b) Absolute Error along TR2

Figure 4.20: Comparison of Pin Power Error for C4V Configuration with other transport calculations (cont.)



(c) Absolute Error along TR3

Figure 4.20: Comparison of Pin Power Error for C4V Configuration with other transport calculations (cont.)

Chapter 5

Conclusion

This marks the end of this report. Given the agreements found in most of the results, the author can justifiably claim success in his preliminary objective, a vanilla implementation of Method Of Characteristics in 2D geometry to solve neutron transport problems.

The code works well with usual uranium assembly. But to achieve excellence, much work is still needed since it provides appreciable error in absorbing medium and MOX assemblies. Incorporation of a higher order P_N approximation for anisotropic scattering is the top priority here, which was the main source of error in all NEA benchmarks. A P_3 approximation is usually adequate for MOX analysis, but a higher order approximation will further improve the fidelity of the code and is the target. A further optimization would be to replace flat source approximation with higher-order source approximation to enable it in handling high flux gradients. This will allow a more accurate solution with fewer meshes.

To reduce the computational effort required, some form of diffusion acceleration method has to be adopted. CMFD(Coarse Mesh Finite Difference) is the preferred choice in this case, which is found to be applied in almost all transport codes now-a-days. Execution speed of the code can also be be boosted by writing transport sweep algorithm in C and couple it to the main module using Cython. This will lead to a significant reduction in execution time. Writing the sweep code in C should now be easier since the main algorithm is available and studied well.

With these modifications, the code will become fully capable of solving the usual rectangular assembly problems. For hexagonal assemblies, a different ray-tracing routine will have to be developed.

The recent trend in lattice codes demands fine-mesh calculation preserving the geometry of fuel rods. Incorporating this will require a different ray tracing routine as well, which renders the routine used here out of trend. So, a further scope to improve this code and catch up with the trend is to extend its capability to heterogeneous geometry as well.

The author intends to carry on adding these features in the near future to realize a robust solver in two dimensions. It is his hope the finished product will find its place in a widely used open source lattice code available for anyone who aspires to be a reactor physicist.

References

[1] S. Igor et al., *The role of High Performance Computing in the Nuclear Energy Sector*, Nuclear Society of Slovenia, Sept. 2015, URL: http://publications.jrc.ec.europa.eu/repository/handle/JRC97395.

[2] M. Y. Tikhonchev et al., *The role of computer simulation in nuclear technologies development* (2001), URL: https://www.osti.gov/etdeweb/servlets/purl/20236667.

[3] J. J. Duderstadt, *Nuclear reactor analysis*, Wiley, 1976, URL: https://deepblue.lib.umich.edu/bitstream/handle/2027.42/89079/1976_Nuclear_Reactor_Analysis.pdf.

[4] Y. Oka, *Nuclear reactor design*, Springer, 2014.

[5] D. Knott and A. Yamamoto, *Lattice Physics Computations*, in: *Handbook of Nuclear Engineering*, ed. by D. G. Cacuci, Springer US, Boston, MA, 2010, pp. 913–1239, ISBN: 978-0-387-98149-9, DOI: 10.1007/978-0-387-98149-9_9, URL: https://doi.org/10.1007/978-0-387-98149-9_9.

[6] B. Kochunas, *A Hybrid Parallel Algorithm for the 3-D Method of Characteristics Solution of the Boltzmann Transport Equation on High Performance Compute Clusters*, PhD thesis, University of Michigan, 2013, URL: https://deepblue.lib.umich.edu/handle/2027.42/100072.

[7] J. Askew, *A characteristics formulation of the neutron transport equation in complicated geometries*, tech. rep. AAEW-M 1108, United Kingdom Atomic Energy Authority, 1972.

[8] M. Halsall, *CACTUS, a characteristics solution to the neutron transport equations in complicated geometries*, tech. rep. AEEW-R 1291, Atomic Energy Establishment, Winfrith, Dorchester, Dorset, UK, 1980.

[9] L. Goldberg, J. Vujic, and A. Leonard, *The characteristics method in general geometry*, Transactions of the American Nuclear Society 73 (1995), pp. 173–174.

[10] N.-Z. Cho and S.-G. Hong, *CRX: a transport theory code for cell and assembly calculations based on characteristic method*, in: *Proceedings of the international conference on physics of reactors, PHYSOR96*, vol. 1, 1996.

[11] R. Roy, *The cyclic characteristics method*, in: *Proc. Int. Topical Meeting on the Physics of Nuclear Science and Technol, Long Island, USA, Oct. 5-8, 1998*, vol. 1, 1998, p. 407.

[12] T. Postma and J. Vujic, *The method of characteristics in general geometry with anisotropic scattering*, in: *Proc. Int. Conf. Mathematics and Computation, Reactor Physics and Environmental Analysis in Nuclear Applications*, vol. 1215, 1999.

[13] S. Kosaka and E. Saji, *Transport theory calculation for a heterogeneous multi-assembly problem by characteristics method with direct neutron path linking technique*, Journal of nuclear science and technology 37.12 (2000), pp. 1015–1023.

[14] K.-S. Kim, *Benchmark calculations of DENT-2D code for PWR fuel assemblies*, in: *Proc. Int. Conf. on the New Frontiers of Nuclear Technology: Reactor Physics, Safety and High-Performance Computing, Seoul, Korea, Oct. 7-10, 2002*, 2002.

[15] T. Jevremovic, T. Ito, and Y. Inaba, *ANEMONA: multiassembly neutron transport modeling*, Annals of Nuclear Energy 29.17 (2002), pp. 2105–2125.

[16] A. Hébert, *Applied reactor physics*, Presses inter Polytechnique, 2009.

[17] G. Marleau, R. Roy, and A. Hébert, *DRAGON: a collision probability transport code for cell and supercell calculations*, Report IGE-157, Institut de génie nucléaire, École Polytechnique de Montréal, Montréal, Québec (1994).

[18] D. Knott, B. H. Forssen, and M. Edenius, *CASMO-4, A Fuel Assembly Burnup Program, Methodology*, Studsvik/SOA-95/2, Studsvik of America, Inc (1995).

[19] N. Sugimura et al., *Neutron transport models of AEGIS: An advanced next-generation neutronics design system*, Nuclear science and engineering 155.2 (2007), pp. 276–289.

[20] D. Knott and E. Wehlage, *Description of the LANCER02 lattice physics code for single-assembly and multibundle analysis*, Nuclear science and engineering 155.3 (2007), pp. 331–354.

[21] C. Wemple et al., *Recent advances in the HELIOS-2 lattice physics code*, in: *Proceedings of the International Conference on Reactor Physics, Nuclear Power: A Sustainable Resource Casino-Kursaal Conference Center*, 2008.

[22] E. Masiello, R. Sanchez, and I. Zmijarevic, *New numerical solution with the method of short characteristics for 2D heterogeneous cartesian cells in the APOLLO2 code: Numerical analysis and tests*, Nuclear Science and Engineering 161.3 (2009), pp. 257–278.

[23] D. She, Z. Liu, and J. Zhao, *The Laputa code for lattice physics analyses*, Annals of Nuclear Energy 75 (2015), pp. 303–308.

[24] H. G. Joo et al., *Methods and performance of a three-dimensional whole-core transport code DeCART* (2004).

[25] W. Boyd et al., *The OpenMOC method of characteristics neutral particle transport code*, Annals of Nuclear Energy 68 (2014), pp. 43–52.

[26] M. D. TEAM et al., *MPACT Theory Manual, 2.2. 0*, Oak Ridge National Laboratory and University of Michigan, CASL-U-2015-0078-000 (2015).

[27] B. Kochunas et al., *Overview of development and design of MPACT: Michigan parallel characteristics transport code*, in: *Proceedings of the 2013 International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering-M and C 2013*, 2013.

[28] A. J. Hoffman and J. C. Lee, *A time-dependent neutron transport method of characteristics formulation with time derivative propagation*, Journal of Computational Physics 307 (2016), pp. 696–714.

[29] G. Bell and S. Glasstone, *Nuclear reactor theory*, Van Nostrand Reinhold Co., 1970.

[30] J. Weisman and A. Tentner, *Application of the method of characteristics to solution of nuclear engineering problems*, Nuclear Science and Engineering 78.1 (1981), pp. 1–29.

[31] *PYPL PopularitY of Programming Language*, URL: `http://pypl.github.io/PYPL.html` (visited on 06/22/2019).

[32] *Scipy Lecture Notes¶*, URL: `https://scipy-lectures.org/`.

[33] G. Zaccone, *Python parallel programming cookbook*, Packt Publishing Ltd, 2015.

[34] N. Kumar, *Multiprocessing in Python: Set 1 (Introduction)*, Feb. 2018, URL: `https://www.geeksforgeeks.org/multiprocessing-python-set-1/`.

[35] R. Modak, D. Sahni, and S. Paranjape, *Evaluation of higher K-eigenvalues of the neutron transport equation by S_n-method*, Annals of Nuclear Energy 22.6 (1995), pp. 359–366.

[36] D. E. Kornreich and D. K. Parsons, *The Green's function method for effective multiplication benchmark calculations in multi-region slab geometry*, Annals of Nuclear Energy 31.13 (2004), pp. 1477–1494.

[37] A. Sood, R. A. Forster, and D. K. Parsons, *Analytical benchmark test set for criticality code verification*, Progress in Nuclear Energy 42.1 (2003), pp. 55–106.

[38] J. Stepanek, T. Auerbach, and W. Hälg, *Calculation of four thermal reactor benchmark problems in XY geometry*, tech. rep., Eidgenoessisches Inst. fuer Reaktorforschung, 1982.

[39] P. S. Brantley and E. W. Larsen, *The simplified P_3 approximation*, Nuclear Science and Engineering 134.1 (2000), pp. 1–21.

[40] L.-Z. Cao and H.-C. Wu, *Spherical harmonics method for neutron transport equation based on unstructured-meshes*, Nucl. Sci. Tech 15.6 (2004), pp. 335–339.

[41] W. Hongchun et al., *Transmission probability method based on triangle meshes for solving unstructured geometry neutron transport problem*, Nuclear engineering and design 237.1 (2007), pp. 28–37.

[42] T. Mazumdar and S. Degweker, *Solution of neutron transport equation by Method of Characteristics*, Annals of Nuclear Energy 77 (2015), pp. 522–535.

[43] J. Wood and M. Williams, *Recent progress in the application of the finite element method to the neutron transport equation*, Progress in Nuclear Energy 14.1 (1984), pp. 21–40.

[44] C. Cavarec et al., *Benchmark calculations of power distribution within assemblies*, tech. rep. NEA/NSC/DOC (94) 28, Nuclear Energy Agency, 1994.

[45] W. Rhoades and R. Childs, *Updated version of the DOT 4 one-and two-dimensional neutron/photon transport code*, tech. rep., Oak Ridge National Lab., TN (USA), 1982.

[46] B. J. Pijlgroms, J. Oppe, and H. Oudshoorn, *The PASC-3 code system and the UNIPASC environment* (1991).

[47] K. Tsuchihashi et al., *Revised SRAC code system/1*, in: *Research on thorium fuel*, 1987.

[48] K. Lathrop and F. Brinkley, *TWOTRAN-II: An interfaced, exportable version of the TWOTRAN code for two-dimensional transport*, tech. rep., Los Alamos Scientific Lab., N. Mex.(USA), 1973.

[49] C. Maeder, *QP 1: a transport program in xy geometry based on function expansions in angle and space*, Eidg. Institut für Reaktorforschung, 1975.

[50] R. Blomquist, *VIM3. 0. Continuous Energy MC Neutron/Photon*, tech. rep., Argonne National Lab., IL (United States), 1991.

[51] W. Bernnat et al., *Physical and computational properties of a multigroup assembly code based on the $J^{\pm}$and P^{ik}-concept*, in: *Advances in Reactor Physics, Mathematics and Computation. Volume 3*, 1987.

[52] R. E. Alcouffe et al., *User Guide for TWODANT: A Code Package for Two-Dimensional, Diffusion-Accelerated, Neutral Particle Transport*, tech. rep., Los Alamos National Laboratory report LA-10049-M (February 1990), 1990.

[53] D. Sato and T. Takeda, *Effect of anisotropic scattering on MOX fuel analysis*, tech. rep., 2000.

[54] C. Yang and P. Benoist, *Scattering anisotropy and neutron leakage in reactor lattices*, Nuclear Science and Engineering 86.1 (1984), pp. 47–62.

Appendix A

Algorithms

Algorithm 1 Generation of ray-tracing data for 2D implementation

```
Input: Mesh Data, Azimuthal Angles $\beta_b^* \ \forall \ b \in B$, ray spacing $d^*$
Compute $\beta_b$, $d_b$ using equations (3.75)-(3.78) $\ \forall \ b \in B$
Compute azimuthal quadrature using equations (3.80)-(3.82)
for all $b \in B$ do
    $\mathbb{L} \leftarrow \emptyset$
    Locate the starting x-coordinates for all rays on lower boundary
    Store the starting points in $\mathbb{L}$
    $k \leftarrow 0$
    for all $x_0$ in $\mathbb{L}$ do
        $k \leftarrow k + 1$
        $x, y \leftarrow x_0, 0$
        repeat
            locate and store mesh $i$
            find mesh intersection point $(x', y')$ along the ray
            calculate and store $l_{b,i,k} = \sqrt{(x' - x)^2 + (y' - y)^2}$ separately for $k$ and $i$.
            $x, y \leftarrow x', y'$
            if $(x, y)$ on boundary then
                Reflect ray
            end if
            if $x \in \mathbb{L} \wedge y = 0$ then
                $\mathbb{L} \leftarrow \mathbb{L} \setminus \{x\}$      ▷ Removing $(x, 0)$ as starting point from the list
                                                    ▷ since it is intersected by the cyclic ray
            end if
        until $x = x_0 \wedge y = 0$
    end for
end for
```

Algorithm 2 Transport Sweep in 1D

```
function 1DSWEEP($\boldsymbol{q}_g, \mathbf{Q}_g, \boldsymbol{\phi}_g$)
    $\Phi_{g,i} \leftarrow \frac{q_i}{\Sigma_t^{g,i}} \;\; \forall \;\; i \in I$
    $i \leftarrow 1$                                          ▷ Initiating ray-tracing from left-end
    Direction←Forward
    while $i > 0$ do
        for  all $b \in B$ do                                  ▷ Loop over all azimuthal directions
            for all $p \in P$ do                                  ▷ Loop over all polar directions
                $s_{b,p,i} \leftarrow \frac{l_i}{\sin\theta_p \, \cos\beta_b}$                    ▷ Equation (3.57)
                $\Delta\phi_{g,b,p,i} \leftarrow \left(\phi_{g,b,p} - \frac{Q_{g,b,p,i}}{\Sigma_t^{g,i}}\right)\left[1 - \exp\left(-\Sigma_t^{g,i} s_{b,p,i}\right)\right]$
                                                                 ▷ Equation (3.62)
                $\Phi_{g,i} \leftarrow \Phi_{g,i} + \frac{\omega_b \omega_p \Delta\phi_{g,b,p,i}}{\Sigma_t^{g,i} \, s_{b,p,i}}$                ▷ Equation (3.63)
                $\phi_{g,b,p} \leftarrow \phi_{g,b,p} - \Delta\phi_{g,b,p,i}$                    ▷ Equation (3.64)
            end for
        end for
        if Direction = Forward then
            $i \leftarrow i + 1$                    ▷ Incrementing mesh position to trace forward
        else if Direction = Backward then
            $i \leftarrow i - 1$
        end if
        if i=I+1 then
            Direction←Backward
            Store $\boldsymbol{\phi}_g$ in $\boldsymbol{\phi}_{g,boun}$                               ▷ To check convergence
            $\boldsymbol{\phi}_g \leftarrow \boldsymbol{\phi}_g \times \alpha_{Right}$                      ▷ Applying Boundary Condition
        else if i=0 then
            Store $\boldsymbol{\phi}_g$ in $\boldsymbol{\phi}_{g,boun}$
            $\boldsymbol{\phi}_g \leftarrow \boldsymbol{\phi}_g \times \alpha_{Left}$
        end if
    end while
    return $\boldsymbol{\Phi}_g, \boldsymbol{\phi}_g, \boldsymbol{\phi}_{g,boun}$
end function
```

Algorithm 3 Transport Sweep in 2D

```
function 2DSWEEP($\boldsymbol{q}_g, \mathbf{Q}_g, \boldsymbol{\phi}_g$)
    $\Phi_{g,i} \leftarrow \frac{q_i}{\Sigma_t^{g,i}} \;\forall\; i \in I$
    for  all $b \in B$ do                                   ▷ Loop over all azimuthal directions
        for all $k \in K$ do                                ▷ Loop over all rays
            $w \leftarrow 1$                                ▷ Ray Index
            $i \leftarrow i_0$                              ▷ Initiates ray from starting mesh
            Direction←Forward
            while $w > 0$ do
                for all $p \in P$ do                        ▷ Loop over all polar directions
                    $\Delta\phi_{g,b,p,i,k} \leftarrow E_{g,b,p,i,k}\left(\phi_{g,b,p,k} - \frac{Q_{g,b,p,i}}{\Sigma_t^{g,i}}\right)$    ▷ Equation (3.86)
                    $\Phi_{g,i} \leftarrow \Phi_{g,i} + \frac{\omega_b\omega_p \sin\theta_p \,\Delta\phi_{g,b,p,i,k}}{\Sigma_t^{g,i} \sum_{k\in i} l_{b,i,k}}$    ▷ Equation (3.87)
                    $\phi_{g,b,p,k} \leftarrow \phi_{g,b,p,k} - \Delta\phi_{g,b,p,i,k}$    ▷ Equation (3.88)
                end for
                $i \leftarrow i'$                           ▷ Locating next mesh
                if Direction = Forward then
                    $w \leftarrow w + 1$                    ▷ Incrementing ray partition to trace forward
                else if Direction = Backward then
                    $w \leftarrow w - 1$
                end if
                if $i$ on boundary then
                    Store $\phi_{g,b,p,k}$ in $\boldsymbol{\phi}_{g,boun}$    ▷ To check convergence
                    $\phi_{g,b,p,k} \leftarrow \phi_{g,b,p,k} \times \alpha_{boun}$    ▷ Applying Boundary Condition
                end if
                if $w = W + 1$ then                         ▷ End of ray $k$
                    Direction←Backward
                end if
            end while
        end for
    end for
    return $\boldsymbol{\Phi}_g, \boldsymbol{\phi}_g, \boldsymbol{\phi}_{g,boun}$
end function
```

Algorithm 4 One-Group Eigenvalue Solver

```
function 1GSOLVER
    $k'_{eff} \leftarrow 1$    ▷ Initiated to preserve values preceding iteration
    $\Phi'_i \leftarrow 1 \ \forall \ i \in I$
    $\phi_{b,p,k} \leftarrow 0 \ \forall \ b,p,k \in \{B,P,K\}$
    $\boldsymbol{\phi}_{boun} \leftarrow 0$
    $\boldsymbol{\phi}'_{boun} \backsim U(0,1)$    ▷ Initiated with random numbers
    $res_k \leftarrow \infty$
    $f'_i \leftarrow \nu\Sigma^i_f\Phi'_i \ \forall \ i \in I$    ▷ Calculation of initial fission source
    while $res_k > tol_k$ do    ▷ Initiating outer iteration
        $q_i \leftarrow \frac{f'_i}{k'_{eff}} + \Sigma^i_s\Phi'_i \ \forall \ i \in I$    ▷ Calculation of total source
        $Q_i \leftarrow \frac{q_i}{4\pi} \ \forall \ i \in I$    ▷ Finding angular Source
        $res_\Phi \leftarrow \infty$
        while $res_\Phi > tol_\Phi$ do    ▷ Initiating inner iteration
            $\mathbf{\Phi}, \boldsymbol{\phi}, \boldsymbol{\phi}_{boun} \leftarrow$ SWEEP$(\boldsymbol{q}, \mathbf{Q}, \boldsymbol{\phi})$    ▷ 1DSweep from Algorithm 2 or
                                                    ▷ 2DSweep from Algorithm 3
            $res_\Phi \leftarrow \max\left(\left\{\max_{i\in I}\left|\frac{\Phi_i - \Phi'_i}{\Phi_i}\right|, \max\left|\frac{\boldsymbol{\phi}_{boun} - \boldsymbol{\phi}'_{boun}}{\boldsymbol{\phi}_{boun}}\right|\right\}\right)$
            $\mathbf{\Phi}' \leftarrow \mathbf{\Phi}$
            $\boldsymbol{\phi}'_{boun} \leftarrow \boldsymbol{\phi}_{boun}$
        end while
        $f_i \leftarrow \nu\Sigma^i_f\Phi'_i \ \forall \ i \in I$
        $k_{eff} \leftarrow k'_{eff} \times \frac{\sum_i^I f_i A_i}{\sum_i^I f'_i A_i}$    ▷ Calculation of eigenvalue
                                                    ▷ $l_i$ instead of $A_i$ in 1D
        $res_k \leftarrow |k_{eff} - k'_{eff}|$
        $k'_{eff} \leftarrow k_{eff}$
        $f'_i \leftarrow f_i \ \forall \ i \in I$
    end while
    return $k_{eff}, \mathbf{\Phi}$
end function
```

Algorithm 5 Multigroup Eigenvalue Solver

```
function MGsolver
    $k'_{eff} \leftarrow 1$    ▷ Initiated to preserve values preceding iteration
    $\Phi'_{g,i} \leftarrow 1 \ \forall \ i \in I$
    $\phi_{g,b,p,k} \leftarrow 0 \ \forall \ g,b,p,k \in \{G,B,P,K\}$
    $\boldsymbol{\phi}_{g,boun} \leftarrow 0 \ \forall \ g \in G$
    $\boldsymbol{\phi}'_{g,boun} \backsim U(0,1) \ \forall \ g \in G$    ▷ Initiated with random numbers
    $res_k \leftarrow \infty$
    $f'_{g,i} \leftarrow \chi_g \sum_g^G \nu\Sigma_f^{g,i}\Phi'_{g,i} \ \forall \ i,g \in \{I,G\}$    ▷ Calculation of initial fission source
    while $res_k > tol_k$ do    ▷ Initiating outer iteration
        $q_{ind}^{g,i} \leftarrow \frac{f'_{g,i}}{k'_{eff}} + \sum_{g',g'\neq g}^{G} \Sigma_s^{g'\to g,i}\Phi'_{g,i} \ \forall \ i,g \in I,G$
            ▷ Calculation of total source
            ▷ excluding in-scattering contributions
        for all $g \in G$ do
            $res_\Phi \leftarrow \infty$
            while $res_\Phi > tol_\Phi$ do    ▷ Initiating inner iteration for group $g$
                $q_{g,i} \leftarrow q_{ind}^{g,i} + \Sigma_s^{g\to g,i}\Phi'_{g,i} \ \forall \ i \in I$    ▷ Calculation of total source
                $Q_{g,i} \leftarrow \frac{q_{g,i}}{4\pi} \ \forall \ i, \in I$    ▷ Finding angular Source
                $\boldsymbol{\Phi}_g, \boldsymbol{\phi}_g, \boldsymbol{\phi}_{g,boun} \leftarrow$SWEEP$(\boldsymbol{q}_g, \mathbf{Q}_g, \boldsymbol{\phi}_g)$    ▷ 1DSweep from Algorithm 2
                    ▷ or 2DSweep from Algorithm 3
                $res_\Phi \leftarrow \max\left(\left\{\max_{i\in I}\left|\frac{\Phi_{g,i}-\Phi'_{g,i}}{\Phi_{g,i}}\right|, \max\left|\frac{\boldsymbol{\phi}_{g,boun}-\boldsymbol{\phi}'_{g,boun}}{\boldsymbol{\phi}_{g,boun}}\right|\right\}\right)$
                $\boldsymbol{\Phi}'_g \leftarrow \boldsymbol{\Phi}_g$
                $\boldsymbol{\phi}'_{g,boun} \leftarrow \boldsymbol{\phi}_{g,boun}$
            end while
        end for
        $f_{g,i} \leftarrow \chi_g \sum_g^G \nu\Sigma_f^{g,i}\Phi'_{g,i} \ \forall \ i,g \in \{I,G\}$
        $k_{eff} \leftarrow k'_{eff} \times \frac{\sum_g^G \sum_i^I f_{g,i}A_i}{\sum_g^G \sum_i^I f'_{g,i}A_i}$    ▷ Calculation of eigenvalue
            ▷ $l_i$ instead of $A_i$ in 1D
        $res_k \leftarrow |k_{eff} - k'_{eff}|$
        $k'_{eff} \leftarrow k_{eff}$
        $f'_{g,i} \leftarrow f_{g,i} \ \forall \ i,g \in I,G$
    end while
    return $k_{eff}, \boldsymbol{\Phi}$
end function
```

Appendix B

Sample Codes

B.1 Sample Input File

```
#This is the input file for Heterogenous Slab Geometry Case 1 intended to be
↪  solved with one group solver in one dimension. The problem is taken from
↪  Kornreich and Parsons(2004)


f_l=1/0.415          #length of fuel section (1 mfp)
r_l=1/0.371          #length of reflector section (1 mfp)


#the whole region consists of seven sections. The length of all the sections
↪  are listed below in the sequence they are arranged. The purpose of this
↪  list is to decide lengths to assign mesh divisions.
X=[r_l,f_l,r_l,f_l,r_l,f_l,r_l]


#The number of mesh divisions corresponding to each of these lengths assigned
↪  are listed below
NX=[125,125,125,125,125,125,125]


#Boundary Conditions
x0=0    #albedo on the left side
xn=0    #albedo on the right side


#Number of azimuthal divisions in 360 degrees
azim_div=64

nu=1 #Neutrons per fission. Assigned 1 because it is given as a product with
↪  fission cross section
```

```python
#Defining geometry
def geometry(x):

    f_l=1/0.415
    r_l=1/0.371

    one=(x>0) and (x<r_l)
    two=(x>r_l) and (x<(f_l+r_l))
    three=(x>(f_l+r_l)) and (x<(f_l+(2*r_l)))
    four=(x>(f_l+(2*r_l))) and (x<((2*f_l)+(2*r_l)))
    five=(x>((2*f_l)+(2*r_l))) and (x<((2*f_l)+(3*r_l)))
    six=(x>((2*f_l)+(3*r_l))) and (x<((3*f_l)+(3*r_l)))
    seven=(x>((3*f_l)+(3*r_l))) and (x<((3*f_l)+(4*r_l)))

    fuel=two or four or six
    reflector=one or three or five or seven

    return fuel,reflector

#Assigning cross sections

def sigma_scatter(x):
    fuel,reflector=geometry(x)
    return (fuel*0.334)+(reflector*0.334)

def sigma_total(x):
    fuel,reflector=geometry(x)
    return (fuel*0.415)+(reflector*0.371)

def sigma_fis(x):
    fuel,reflector=geometry(x)
    return (fuel*0.178)+(reflector*0)
```

B.2 Sample Solver Code

```python
#MOC Solver for one-group eigenvalue problem in one dimension

def moc1g1D():

    import math, numpy as np #importing necessary modules

    #importing computational parameters and material data from input file
    from input_file import X ,NX ,x0 ,xn ,azim_div , nu ,sigma_scatter
    ↪  ,sigma_total ,sigma_fis
    #computing length of each mesh and storing them
    delx=np.array([])
    N=sum(NX)
    for i in range(len(X)):
        delx=np.append(delx,np.ones(NX[i])*X[i]/NX[i])

    #Tabuchi-Yamamoto Polar Quadrature Set
    sintheta_j=np.array([0.166648,0.537707,0.932954])
    omega_j=np.array([0.046233,0.283619,0.670148])*2

    #computing azimuthal directions and quadrature
    azim=np.arange(np.pi/azim_div,np.pi/2,2*np.pi/azim_div)
    omega_azim=4*math.pi/azim_div

    phi=np.ones(N)          #array to store scalar flux
    phi_last=phi.copy()     #array to store scalar flux from preceding inner
                            #iteration
    phi_j=np.zeros((len(azim),3))          #array to store angular flux
    phi_j_boun=np.zeros((2,len(azim),3))   #array to store angular flux
                                           #on boundary
    #array to store angular flux on boundary from preceding inner iteration
    phi_j_last=np.random.randint(1,10,np.shape(phi_j_boun))/10

    keff_last=1         #eigenvalue from preceding outer iteration

    #Creating arrays for storing cross-section data
    Sigma_total=np.ones(N)
    Sigma_scatter=np.ones(N)
    Sigma_fis=np.ones(N)

    #Calling cross-section data from the imported functions
    for i in range (N):
        x=np.sum(delx[0:i])+(0.5*delx[i])
        Sigma_total[i]=sigma_total(x)
        Sigma_scatter[i]=sigma_scatter(x)
        Sigma_fis[i]=sigma_fis(x)

    F_source=nu*Sigma_fis*phi         #Computing fission source

    iteration=0     #outer iteration counter
    sweepnumber=0   #inner iteration counter

    while True:     #commencing outer iteration loop

        iteration+=1      #incrementing outer iteration
        print(iteration)
        q=F_source/keff_last+(Sigma_scatter*phi)    #computing total source
        Q=q/(4*math.pi)                             #computing angular source
        while True: #commencing inner iteration loop

            sweepnumber+=1      #incrementing inner iteration

            #initiating transport sweep
            phi=q/Sigma_total   #initiating flux values
            pos=0               #initiating ray-tracing from left-end
            reverse=False       #setting direction to be forward

            while True:         #initiating ray-tracing

                for angle in range(len(azim)):#looping over
                                              #azimuthal divisions
                    for j in range(3):        #looping over polar divisions

                        s=delx[pos]/sintheta_j[j]/math.cos(azim[angle])

                        del_phi=(phi_j[angle,j]-(Q[pos]/Sigma_total[pos]))*
                        ↪   (1-math.exp(-s* Sigma_total[pos]))

                        phi[pos]+= omega_j[j] * omega_azim * del_phi /
                        ↪   (s*Sigma_total[pos])

                        phi_j[angle,j]-=del_phi

                pos+=(-1)**reverse  #increment/decrement mesh position

                if pos==N:          #identifying boundary on right

                    reverse=True        #changing ray-tracing diretion
                    pos-=1              #decrementing mesh position
                    phi_j_boun[0]=phi_j #storing angular flux on boundary
                    phi_j*=xn           #applying boundary condition

                elif pos==-1:       #identifying boundary on left

                    phi_j_boun[1]=phi_j
                    phi_j*=x0

                    break                #stopping ray-tracing

            #transport sweep completed
            #computing residue
            res=np.array([np.max(abs(phi_j_boun-phi_j_last)/phi_j_boun),
            ↪   np.max(abs(phi-phi_last)/phi)])

            if np.max(res)<1e-8: #comparing residue for convergence
                break            #stopping inner iteration
```

```python
            else:       #copying  data for residue calculation in next sweep
                phi_j_last=phi_j_boun.copy()
                phi_last=phi.copy()

        #computing new fission source
        f_source=nu*Sigma_fis*phi

        #computing new eigenvalue
        keff=np.sum(f_source*delx)*keff_last/np.sum(F_source*delx)
        print(keff)

        if abs(keff-keff_last)<1e-6:    #checking convergence of eigenvalue
            break                       #stopping outer iteration

        else:

            keff_last=keff           #updating eigenvalue
            phi_last=phi.copy()      #updating preceding flux
            F_source=f_source.copy() #updating fission source

    return keff,phi          #return statement

#calling the solver
keff,phi=moc1g1D()
```