POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Development of a Python library for the generation of 2D unstructured geometries of radial reflectors in support of industrial model verification

FABIO INZIRILLO
Département de génie physique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie énergétique option nucléaire

Décembre 2023

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Development of a Python library for the generation of 2D unstructured geometries of radial reflectors in support of industrial model verification

présenté par **Fabio INZIRILLO**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

ACKNOWLEDGEMENTS

I am profoundly grateful to Alain Hébert, my research director, for providing me the opportunity to undertake my research project at Framatome and for his unwavering support and assistance throughout my academic journey.

I also wish to express my sincere gratitude to Guy Marleau, my co-director, for our weekly meetings where he introduced me to DRAGON and lattice codes, which have been significantly helpful during my intership work. I would like to thank him for his meticulous attention to detail and support during the review of my thesis.

I also would like to thank prof. Antonio Cammi, who introduced me to neutronics. It was his lessons that inspired me to focus my thesis on reactor physics. I am grateful for his willingness to oversee my studies from Milano.

My supervisors at Framatome, Barbara Vezzoni, Alberto Brighenti, and Laurent Graziano, have provided continous guidance and support throughout my work. I am grateful for their reviews and suggestions for my thesis, and for giving me the opportunity to contribute to a scientific paper. A special thank Barbara for her help with the physical aspects of my work, for coordinating my efforts and continously motivating me with her positive attitude, to Alberto for his invaluable programming advices, and to Laurent for his support with APOLLO3® and SCIENCE. I have learned a lot from you and I hope I will in the future.

I extend my thanks to the entire team at DTIPD, thank to them I have worked in a convivial and stimulating environment.

My gratitude also goes to the team at CEA for their help with the APOLLO3® and TRIPOLI4® issues. A special mention to Simone Santandrea for his *fantastic* support with the APOLLO3® code.

Lastly, I am thankful also to Marco Tiberga and Adrien Willien for sharing their simulation results for the KAIST-like core.

RÉSUMÉ

Les codes utilisés pour les applications industrielles s'appuient sur des schémas de calcul précis et efficaces pour répondre aux besoins et contraintes des applications réelles. Dans le cas du réflecteur radial, des modèles unidimensionnels simplifiés sont généralement adoptés pour calculer les sections efficaces macroscopiques (XS) du réflecteur pour les calculs du cœur. Ces modèles simplifiés peuvent être réexaminés pour améliorer la prédiction de la réponse du cœur lorsque des réflecteurs radiaux lourds sont utilisés, comme dans le cas des types de réacteurs VVER-1000 et EPR. Des modèles de réflecteur radial à deux dimensions peuvent alors être développés et étudiés. Le travail présenté s'inscrit dans ce cadre, contribuant à la mise en place d'une modélisation de réflecteur radial 2D via le développement d'une bibliothèque Python qui permet de générer des géométries et des maillages de ce réflecteur pour les calculs de réseau APOLLO3®, le code déterministe français de nouvelle génération. Pour accomplir cette tâche, la bibliothèque Python a été conçue pour envelopper l'API du générateur de maillage ALAMOS, développée au CEA. ALAMOS permet de générer des géométries APOLLO3® non structurées, fournissant aux utilisateurs des fonctionnalités de haut niveau pour produire des géométries de réflecteur radial. L'outil proposé est en effet destiné à être intégré dans la plateforme NEMESI, le prototype de générateur de bibliothèques multiparamètrées développé dans le cadre du projet H2020 CAMIVVER et orienté vers l'industrialisation du code APOLLO3®. La bibliothèque Python est appliquée à trois études de cas représentatifs de la conception de VVER et des réacteurs à eau pressurisée occidentaux. La capacité de la bibliothèque à fournir des géométries adaptées aux calculs de transport de référence du cœur complet 2D est soulignée, ainsi que les développents futurs. En particulier, deux configurations définies dans le projet H2020 CAMIVVER ont été considérées : le cœur de type KAIST, un petit cœur hétérogène représentatif d'un cas basé sur un REP équipé d'un réflecteur lourd, et le mini-cœur Kozloduy-6, un cas de sept assemblages de combustible représentatif d'une configuration VVER. Une application industrielle à grande échelle est présentée en utilisant un cœur de type EPR. Les résultats sont présentés pour tous les cas avec des comparaisons impliquant des calculs de référence Monte Carlo et des plateformes industrielles lorsque cela est possible.

ABSTRACT

The codes used for industrial applications rely on accurate and efficient calculation schemes to cope with the needs and constraints of real applications. In the case of the radial reflector, simplified one-dimensional models are typically adopted to compute macroscopic reflector cross-sections (XS) for core calculations. These simplified models may be reconsidered to improve the core response prediction when heavy radial reflectors are employed, such as in the case of the VVER-1000 and EPR reactor types. Two-dimensional radial reflector models may then be developed and investigated. The present work is inserted in this framework, contributing to the setup of 2D radial reflector modeling via the development of a Python library that allows to generate such geometries and meshes for APOLLO3® lattice calculations, the new generation French deterministic code. To accomplish this task, the Python library has been conceived to wrap the ALAMOS mesh generator API. ALAMOS is developed at CEA to generate unstructured APOLLO3® geometries, providing the users with high-level functionalities to produce radial reflector geometries. The proposed tool is intended indeed to be integrated into the NEMESI platform, the multiparameter lattice generator prototype developed within the H2020 CAMIVVER project and oriented towards the industrialization of the APOLLO3® code. The Python library is described by presenting the applications to three cases representatives of VVER and western PWR design. The library's capability to provide geometries suitable for 2D full core reference transport calculations is underlined, as well as further possible evolutions. In particular, two configurations defined in the CAMIVVER project have been considered: the KAIST-like core, a small heterogeneous core representative of a PWR-based case equipped with a heavy reflector, and the Kozloduy-6 mini-core, a seven-fuel assembly case representative of a VVER configuration. A full-scale industrial application is provided using an EPR-like core. Results are presented for all cases, with benchmarks against reference Monte Carlo calculations and industrial platforms shown when possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

EPR	European Pressurized Reactor
VVER	Vodo-Vodyanoi Enyergeticheskiy Reaktor
KZL6	Kozloduy-6
API	Application Programming Interface
CEA	Commissariat à l'Énergie Atomique et aux Énergies Alternatives
EDF	Electricité de France
HZP	Hot Zero Power
KAIST	Korea Advanced Institute of Science and Technology
CAMIVVER	Codes And Methods Improvements for VVER comprehensive safety assessment
SMR	Small Modular Reactor
SERMA	Service d'Études de Réacteurs et de Mathématiques Appliquées
BWR	Boiling Water Reactor
SPH	superhomogénéisation

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Industrial applications in the nuclear field rely on high-efficiency validated solvers to simulate the complex behavior of reactors to ensure an efficient operation and comply with independent authorities' safety standards [10]. Given the large number of simulations to run, a trade-off usually emerges between computation time and accuracy to fit the industry's need to produce accurate results in a reasonable amount of time.

To optimize the core performances, new reactor designs are getting increasingly complex: reactor sizes largely vary (from EPR to SMR types), and burnable absorbers are introduced to compensate for reactivity variation during the cycle. Additionally, heavy reflectors are incorporated in the new Generation III (i.e., EPR) designs to improve the neutron economy. Heavy reflectors also have higher absorption capabilities than light ones, offering better vessel protection and increasing the reactor's lifetime. The reactor's complexity, combined with the safety criteria becoming more and more stringent, requires continuous improvement of the available schemes and solvers.

Considering industrial schemes, SCIENCE [11] is currently employed at Framatome. It is based on APOLLO2 [12] for 2D lattice calculations and SMART for 3D core calculations. Reflector cross sections are produced using the Baff-Refl [9] one-dimensional nodal equivalence technique. In heavy reflectors, water constitutes only a small portion of the total volume. Given the reduced moderating effect, the neutron spectrum redirected towards the core is much faster compared to the one resulting from a light reflector. As a result, the neutrons scattered back will penetrate deep into the core, thereby influencing the neutron flux in the innermost regions. While inaccuracies in modeling light reflectors mainly affect the outermost pins of the core, for heavy reflectors, inaccurate modeling could introduce discrepancies across larger core areas.

Developing more accurate models, such as a 2D implementation of Baff-Refl [9], requires a detailed description of the system geometry. Meshers have long been used to produce complex geometries to feed calculation schemes. Among those, ALAMOS [13] was developed by CEA and designed for APOLLO3® [14], a multi-purpose lattice code currently used at Framatome for R&D purposes. Having in mind industrial user's needs, high-level Python libraries to pilot ALAMOS API have been recently developed at Framatome in the framework of the H2020 CAMIVVER project [15], allowing the users to produce in a fast and modular way detailed geometries and meshes of western PWR and VVER assemblies to be used for APOLLO3® calculations.

The present work is inserted in this framework, contributing to the setup of 2D radial reflector modeling via the development of a Python library that allows to generate 2D radial reflector geometries and meshes for APOLLO3® lattice calculations, the new generation French deterministic code. To accomplish this task, the Python library has been conceived to wrap the ALAMOS mesh generator API, developed at CEA to generate unstructured APOLLO3® geometries, providing the users with high-level functionalities to ease the generation of 2D radial reactor geometries to feed calculation schemes and reference calculations. The present work contributes to the analysis of the ALAMOS capabilities to support the industrialization of APOLLO3® code. The work is restricted to the development of the reflector library and the set of a series of reference calculations as preconditions for advanced reflector modeling. In particular, lattice and reflector libraries are applied to produce reference deterministic calculations to support the analysis of APOLLO3® industrialization. Three test cases have been chosen. Two configurations defined in the CAMIVVER project have been considered: the KAIST-like core, a small heterogeneous core representative of a PWR-based case equipped with a heavy reflector serving as a first testing of the library, and the Kozloduy-6 mini-core, a seven-fuel assembly case representative of a VVER configuration. The third case is representative of a full-scale industrial case: an EPR-like core. APOLLO3® calculations were performed in this case and compared to industrial solutions from the SCIENCE platform. In addition, the TRIAGE module available in APOLLO3® has been tested to generate TRIPOLI4® [16] reference Monte Carlo calculations. With this functionality, it has been possible to support the analyses via comparisons against reference Monte Carlo calculations.

1.1 Overview of the used softwares

This section presents the tools used and introduces the work context. A detailed introduction of the ALAMOS mesh generator is provided given its strong link with the current work. The APOLLO3® code is also presented, focusing on the related H2020 CAMIVVER activities oriented towards its industrialization to present the contex this work is inserted in. Other codes, TRIPOLI4® and SCIENCE, are also mentioned in the work.

TRIPOLI4® is the fourth generation of the continuous-energy radiation transport Monte Carlo code developed by the Service d'Études des Réacteurs et de Mathématiques Appliquées (SERMA) at CEA/Saclay [16]. It is used in this work to produce reference Monte Carlo calculations to compare the results obtained in Chapter 5. In the present work, TRIPOLI4® is driven using the TRIAGE tool implemented in APOLLO3®, which allows to automatically generate a TRIPOLI4® input from an APOLLO3® input.

SCIENCE is instead the industrial nuclear code package developed in the 90's by Framatome and currently used for nuclear design purposes. It provides high-efficiency physical models that suits industrial requirements of high accuracy with short calculation time. SCIENCE is piloted through a graphical user interface called COPILOTE, which manages all tasks needed for the calculation, from data input to result processing. SCIENCE implements the classical "two-step approach" to perform its calculations. Lattice calculations are accounted by APOLLO2-F, the Framatome version of the APOLLO2 code developed by CEA, which runs assembly calculations using a 99 neutron-energy group library. Core calculations are solved with a 2-energy-groups nodal expansion approach implemented in SMART [11]. SCIENCE results are used in Paragraph 5.3.3 for a comparison between industrial and R&D codes.

1.1.1 ALAMOS

Dedicated tools have been developed to assist the user with the preparation of input data, including geometry description, since earlier releases of the lattice codes. ALAMOS [13] is a module of the platform SALOME [17], developed by CEA, offering a Python programming interface that can be used to create geometry files for calculations suitable for APOLLO3® [14] and TRIPOLI4® [16].

ALAMOS supports two types of geometry representation: structured and unstructured. The first type is based on geometrical primitive forms (i.e., squared-based lattice) using a parametric approach to describe their attributes. In contrast, the second type is a collection of points and edges (line segments, circles and arcs of circle) drawn in the plane. Unstructured geometries are recommended to draw complex patterns, such as those currently found in research reactors, or complex assembly designs, such as BWR ones. This is also the case of several complex radial reflector configurations found in current operating reactors, like the one found in VVER-1000 [6]. The desired geometry is sketched in objects called layers upon which various properties can be assigned.

ALAMOS layers

ALAMOS is based on a main Python object called `Layer` containing all the information needed for building and exporting geometry. A layer is composed of:

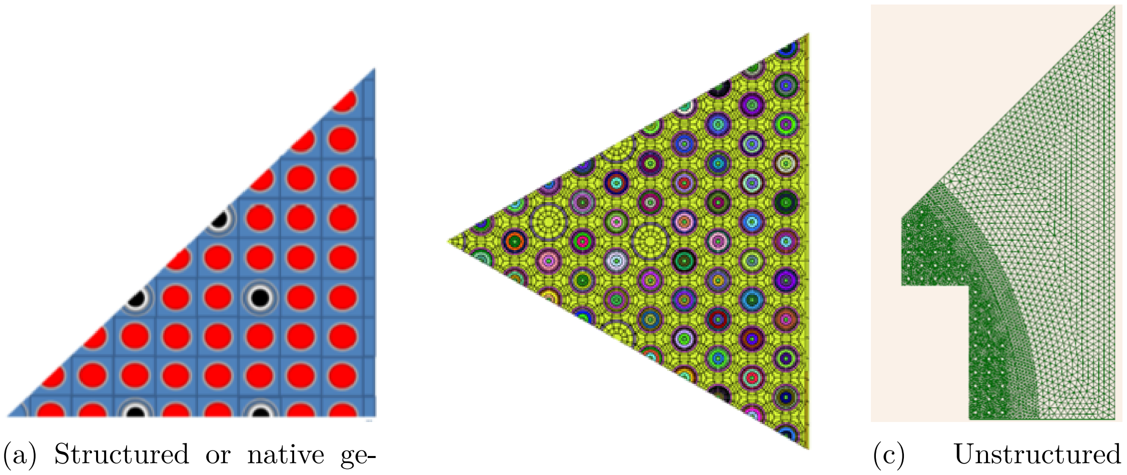
- A geometry
- A set of properties to be assigned to each geometry region called fields (typically names of materials, temperatures).

In order to satisfy user's needs, different kinds of layers have been implemented in ALAMOS. The most frequently used are presented below.

Structured layer This class defines geometry elements using the native geometry description of APOLLO3®, as shown in Figure 1.1a. Structured layers describe basic shapes by means of parametric equations.

Structured layers are limited to simple geometries already accessible using native geometries available in most lattice codes. Structured or native geometries are not optimal for modeling complex structures like radial reflectors. For this reason, the main potentiality of the ALAMOS mesher comes from its ability to deal with unstructured geometries.

Unstructured Layer The class for unstructured layers is called `Layer` and is designed to build unstructured geometries. This type of geometry is described in ALAMOS by a combination of points, segments, arcs and cells. A comparison between structured and unstructured geometries is given in Figure 1.1.



(a) Structured or native geometry of an eighth of PWR assembly

(b) Unstructured geometry of a sixth of VVER assembly

(c) Unstructured geometry mesh of a reflector

Figure 1.1 Examples of structured and unstructured geometries.

Closed regions that are not crossed by any line or arc are called meta-cells. Layer field values can be assigned individually to each meta-cell. Hence, each meta-cell can store different information related to materials, temperature, and other properties used, for instance, for self-shielding/depletion purposes. An example is provided in the following Python script, that creates a `Layer` object that is represented in Figure 1.2:

```
from alamos_engine.layer import Layer

layer_1 = Layer()
layer.createField("materials", 3)
layer_1.drawBox(center_X=0, center_Y=0, len_X=3, len_Y=3)
layer_1.setFieldAtMetaCell(field_name="materials", cell_id=0, val="water_MAT")
layer_1.drawCircle(center_X=0, center_Y=0, radius=1)
layer_1.setFieldAtMetaCell(field_name="materials", cell_id=1, val="clad_MAT")
layer_1.drawCircle(center_X=0, center_Y=0, radius=0.8)
layer_1.setFieldAtMetaCell(field_name="materials", cell_id=2, val="fuel_MAT")
```

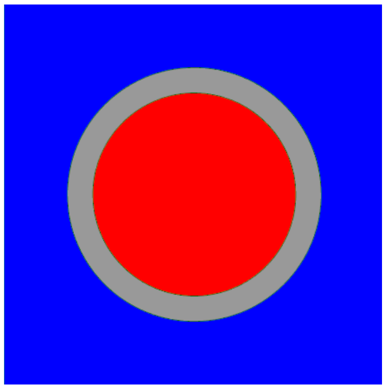


Figure 1.2 Geometry resulting from execution of the previous code

The previous script creates a `Layer` object having a square centered in $(0, 0)$ of side 3 and two concentric circles of radius 1 and 0.8. In ALAMOS, units of measure are dimensionless. The layer also provides information related to the *material* field in each region: the materials of the outermost, central, and innermost region are identified by "water_MAT", "clad_MAT" and "fuel_MAT" fields, respectively. The geometry can be refined by drawing some extra annular regions or by adding sectors by tracing lines over the geometry. Alternatively, some built-in ALAMOS functions allow to automatically refine a meta-cell with a triangular or rectangular mesh.

Furthermore, several geometric operations are available to build more complex geometries. Unstructured layers can overlap, merge, or intersect with other ones. The most used geometric operation, called "referencing", refers to the possibility of inserting a layer inside a meta-cell of another existing layer. The layer to insert is assigned by means of a reference between the cell and the layer itself.

Before exporting the layer, the reference relationship is resolved by inserting the referenced layers in their hosting cells. This process is called dereferenciation (see Figure 1.3).

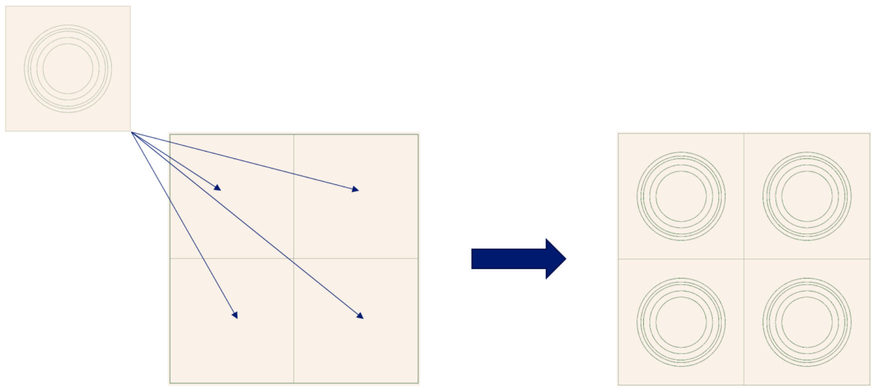


Figure 1.3 Example of a deferenciation procedure

Complex geometries can also be drawn by using a Sketcher object. The Sketcher class is inspired by the tool of the same name available in the Geometry module of SALOME. It allows to build a 2D sketch from an initial point and the successive addition of other points.

ALAMOS lattice library: `NEMESIGeo_C`

ALAMOS is a handy tool for representing complex geometries, providing relatively low-level methods to draw basic shapes to be assembled together. However, the direct use of this software in an industrial framework is limited by the length and complexity of the scripts to be written when reactor components are modeled. Industry requires users to produce rapidly and easily several reactor component geometries using more straightforward tools that do not require knowledge of all the complex ALAMOS features.

In this context, Framatome recently developed a Python library, inspired by a previous work of CEA, called `NEMESIGeo_C` aimed to automate the building procedure of assemblies and cores for both PWR and VVER geometries. The user only needs to define some high-level classes to build objects, such as fuel pins, assemblies, and cores.

The library presented in Chapter 4, called `NEMESIGeo_R`, completes the overall picture, being a tool capable of representing geometries of reflectors used in western PWR (with light and heavy reflectors such as EPR) and VVER.

1.1.2 APOLLO3® code

APOLLO3® is the new generation code for lattice and core calculations developed by CEA with the strong support of Framatome and EDF. The present work focuses on the multigroup lattice code part, which extends the capabilities of the APOLLO2 code, introducing parallelized algorithms using OpenMP directives and 3D characteristics solvers, together with new and improved self-shielding techniques [18]. APOLLO3® is currently used in Framatome for R&D purposes even though the first steps towards its industrialization have been recently taken.

1.1.3 Towards APOLLO3® industrialization: the NEMESI platform

Within the framework of the HORIZON 2020 CAMIVVER project [15], aimed to develop industrial codes and methods for VVER and PWR systems, a prototype of an industrial multi-parameter lattice generator, NEMESI [2], was developed, offering a flexible tool to pilot APOLLO3® for lattice calculations. Figure 1.4 shows the structure of the NEMESI platform. The frontend is based on custom Python modules that wrap the APOLLO3® Python interface and allow to manipulate APOLLO3® backend objects, hence offering the capability to interact with the simulations (i.e., providing access to the data, letting the user introspect the workflow and safely employ backend functionalities). At the end of the simulation, NEMESI retrieves a multi-parameter object (MPO), storing all the required data to run core calculations.

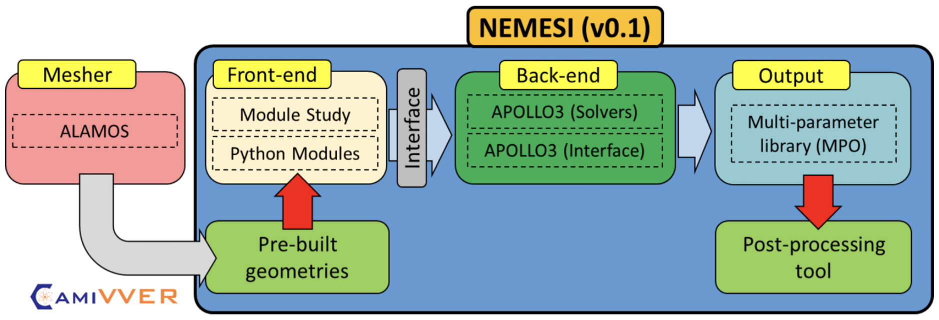


Figure 1.4 Current structure of NEMESI [2]

At the current state, NEMESI uses ALAMOS as an external mesher, relying on pre-built geometries to run its simulations. However, to take a step further in the industrialization process, the lattice libraries presented in Paragraph 1.1.1 and the reflector library developed in the present work will be integrated into the platform for on-the-fly generation of cartesian and hexagonal core configurations. A scheme resuming the planned structure for NEMESI is given in Figure 1.5, where the mentioned `NEMESIGeo_R` is developed in the present work.

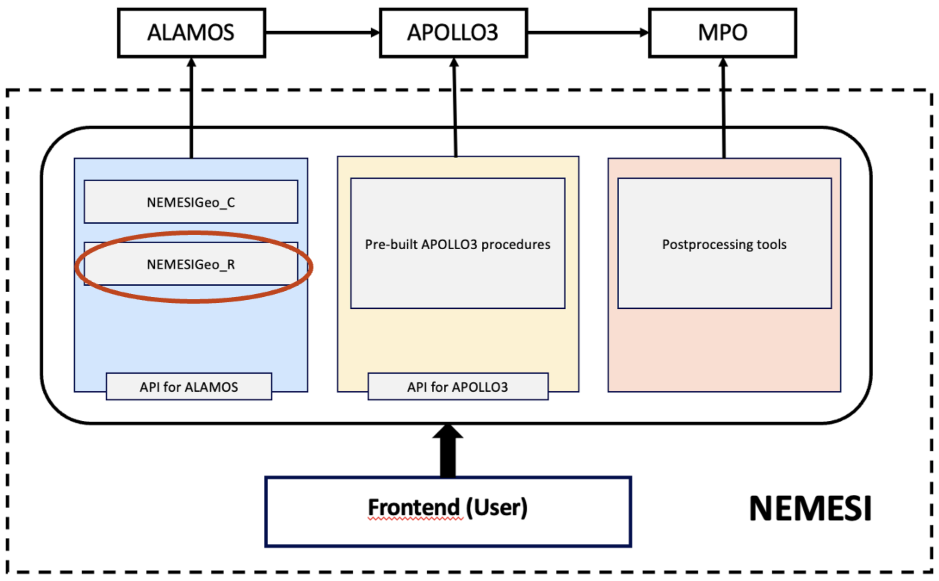


Figure 1.5 Planned structure of NEMESI

1.2 Structure of the work

The work is presented as follows:

- Chapter 2 discusses the main theoretical aspects of neutron transport with the aim to introduce the Collision Probability method and the Method of Characteristics, used in the applications presented in Chapter 5.
- Chapter 3 presents the general structure of a lattice code, such as APOLLO3®, and its most important procedures.
- Chapter 4 describes the reflector library developed in the present work following the logical path that brought its development.
- Chapter 5 shows applications of the library for reference 2D deterministic calculations as a first step to develop an advanced reflector model.
- Chapter 6 draws up conclusions and future perspectives.

CHAPTER 2 Elements of neutron transport theory

This chapter aims to present the main fundamental elements of transport theory. The essential concepts and quantities are first outlined, encompassing cross sections, neutron flux and current, and discussing isotopic evolution. These concepts lay the groundwork for introducing the neutron transport equation, which will be discussed through its different formulations including the differential, integral, characteristic, and steady-state forms. More details can be found in [4].

Subsequently, the most common numerical methods employed to solve the steady-state neutron transport equation are presented. Special attention is given to the methods exploited in this work, namely the collision probability method and the method of characteristics. For the sake of completeness, the spherical harmonics and discrete ordinate methods are also briefly presented.

2.1 Fundamental concepts

2.1.1 Cross sections

The most fundamental quantities required to perform reactor calculation are the cross sections. In reactor physics, the quantity of interest, is the macroscopic cross section $\Sigma_i(\boldsymbol{r}, E, t)$, representing the probability per unit path length for a neutron with energy E in position $\boldsymbol{r}$ to interact with the isotope i. Macroscopic cross sections are defined for any interaction that can occur. These are grouped in scattering and absorption cross sections: $\Sigma_{s,i}(\boldsymbol{r}, E, t)$ and $\Sigma_{a,i}(\boldsymbol{r}, E, t)$. The total macroscopic cross section is the sum of all the cross sections for any reaction x.

$$\Sigma_i(\boldsymbol{r}, E, t) = \Sigma_{s,i}(\boldsymbol{r}, E, t) + \Sigma_{a,i}(\boldsymbol{r}, E, t) = \sum_x \Sigma_{x,i}(\boldsymbol{r}, E, t) \tag{2.1}$$

Each isotope is characterized by its own set of cross sections. For a given medium, the resulting total macroscopic cross section is given by the sum of all the macroscopic cross sections coming from the isotopes present in the medium.

$$\Sigma(\boldsymbol{r}, E, t) = \sum_{i=1}^{N} \Sigma_i(\boldsymbol{r}, E, t) \tag{2.2}$$

Where N is the total number of isotopes in the medium. Macroscopic cross sections are defined starting from a more fundamental quantity, the microscopic cross section, that represents the effective surface of the nucleus for the given reaction.

$$\Sigma_{x,i} = N_i \sigma_{x,i} \tag{2.3}$$

Where N_i is the atomic number density of isotope i. Microscopic cross section data are evaluated and stored in libraries and are used as input files for reactor calculations. The data libraries JEFF 3.1.1 [19] and JEF 2.2 [20] are used in the present work.

2.1.2 Neutron flux and current

The neutron behavior is represented by the motion of a classical particle that can be identified by a vector position $\boldsymbol{r}$, traveling in direction $\boldsymbol{\Omega}$ with speed v at time t. The velocity module is replaced by the kinetic energy E of the neutron given by $E = \frac{1}{2}mv^2$. The neutron population is represented as a distribution, the population density $n(\boldsymbol{r}, E, \boldsymbol{\Omega}, t)$, defined in such a way that $n(\boldsymbol{r}, E, \boldsymbol{\Omega}, t)d^3r dE d^2\Omega$ represents the expected number of neutrons in d^3r about $\boldsymbol{r}$, with energy dE about E and moving in direction $\boldsymbol{\Omega}$ in solid angle $d^2\Omega$ at time t. The angular and integrated fluxes are defined by:

$$\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) = vn(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) \tag{2.4}$$

$$\phi(\boldsymbol{r}, E, t) = \int_{4\pi} d^2\Omega \phi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) \tag{2.5}$$

The neutron flux is the main dependent variable used in the transport equation, and its value is strictly related to the reaction rate. In fact, the angular interaction rate for a given reaction of cross section $\Sigma(\boldsymbol{r}, E)$ is given by:

$$f(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) = \Sigma(\boldsymbol{r}, E)\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) \tag{2.6}$$

We define also the angular and integrated current:

$$\boldsymbol{J}(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) = \boldsymbol{\Omega}\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) \tag{2.7}$$

$$\boldsymbol{J}(\boldsymbol{r}, E, t) = \int_{4\pi} d^2\Omega \boldsymbol{J}(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) \tag{2.8}$$

2.1.3 Isotopic evolution

Nuclear reactions occurring in the reactor lead to an evolution of the isotopic composition of media. Accordingly, it is expected that cross sections will vary with time due to this change in composition.

The rate of change of the density for isotope k is given by the sum of a source term and a leakage term. There is a certain probability, expressed in terms of a yield $Y_{k,l,x}$, that the isotope k is produced after a reaction x from isotope l. Moreover, isotope k can be formed following a radioactive decay of an unstable isotope u. Accordingly, we can express the source term associated to the rate of production of isotope k as:

$$S_k(t) = \sum_x \sum_l Y_{k,l,x} < \Sigma_{x,l}(t)\phi(t) > + \sum_l m_{k,u}(t) N_u(t) \tag{2.9}$$

$$< \Sigma_{f,l}(t)\phi(t) >= N_l \int_0^\infty dE\ \sigma_{f,l}(E,t)\phi(t,E) \tag{2.10}$$

Where $m_{k,u}(t)$ is the decay constant of isotope u for the production of isotope k. The leakage term comes from the fact that isotope k may be an unstable element and hence can decay into another isotope. Moreover, it can absorb a neutron and transform into a different isotope. The rate of loss for the isotope k is then given by:

$$\Lambda_k(t)N_k(t) = \lambda_k N_k(t) + N_k < \sigma_{a,k}(t)\phi(t) > \tag{2.11}$$

Where λ_k is the radioactive decay constant of isotope k. Finally, we can express the rate of change in the isotopic density for the isotope k as:

$$\frac{dN_k}{dt} + \Lambda_k(t)N_k(t) = S_k(t), \quad k = 1, \dots, K \tag{2.12}$$

These equations, also known as the Bateman equations, predict the isotopic composition of mixtures in the reactor. It should be noted that the equation requires knowledge of the neutron flux in order to solve them. Depletion effects on macroscopic cross sections are, for normal operation conditions, relatively slow and take some relatively long time before manifesting their effects. For this reason, even when considering time-dependent problems, it is customary to consider the macroscopic cross sections to depend only on the position and energy. Variation of cross section with time is accounted in the so-called burnup calculations, where isotopic evolution is computed over a time interval assuming a constant power.

2.2 Neutron Transport Equation

The neutron transport equation is a mathematical model describing the behavior of neutrons in a nuclear reactor or other systems. It accounts for production, scattering, absorption and leakage of neutrons, as well as their energy and angular distribution. Solving the neutron transport equation is essential for predicting the power distribution in the core, performing criticality calculations and safety analysis of nuclear systems.

2.2.1 Differential formulation

The rate of change of the neutron population $d^3\mathcal{A}$ in a control volume C is given by the contribution of different terms. Neutrons stream in and out of the boundaries of C at a rate $d^3\mathcal{B}$, and their rate of collisions within C is given by $d^3\mathcal{C}$.

$$\begin{aligned} d^3\mathcal{B} &= \int_{\partial C} d^2r(\boldsymbol{N}\cdot\boldsymbol{\Omega})\phi(\boldsymbol{r},E,\boldsymbol{\Omega},t)dEd^2\Omega && (2.13)\\ &= \int_C d^3r\boldsymbol{\nabla}\cdot\boldsymbol{\Omega}\phi(\boldsymbol{r},E,\boldsymbol{\Omega},t)dEd^2\Omega && (2.14)\\ d^3\mathcal{C} &= \int_C d^3rf(\boldsymbol{r},E,\boldsymbol{\Omega},t)dEd^2\Omega && (2.15)\\ &= \int_C d^3r\Sigma(\boldsymbol{r},E)\phi(\boldsymbol{r},E,\boldsymbol{\Omega},t)dEd^2\Omega && (2.16)\end{aligned}$$

Here $\boldsymbol{N}$ is the unit vector normal to the boundaries of C, pointing outwards. Accordingly, $d^3\mathcal{B}$ will be positive if the outgoing neutrons prevails over those entering $\mathcal{C}$. Neutrons are generated in C by fission reactions and scattering collisions. The two terms are usually collected in a source term Q and the rate of production of neutrons $d^3\mathcal{D}$ is given by:

$$\begin{aligned} d^3\mathcal{D} &= \int_C d^3rQ(\boldsymbol{r},E,\boldsymbol{\Omega},t)dEd^2\Omega && (2.17)\\ Q(\boldsymbol{r},E,\boldsymbol{\Omega},t) &= \int_{4\pi} d^2\Omega' \int_0^\infty dE'\Sigma_s(\boldsymbol{r},E\leftarrow E',\boldsymbol{\Omega}\leftarrow\boldsymbol{\Omega'})\phi(\boldsymbol{r},E',\boldsymbol{\Omega'},t)\\ &+\frac{1}{4\pi}\sum_{j=1}^{J}\chi_j(E)\int_0^\infty dE'\nu\Sigma_{f,j}(\boldsymbol{r},E')\phi(\boldsymbol{r},E',\boldsymbol{\Omega},t) && (2.18)\end{aligned}$$

Where fission term is considered to be isotropic, J is the total number of fissionable isotopes, and $\chi(E)$ is the distribution probability for a fission neutron to have an energy E within a dE interval. Note that in the definition of Q, eventual external sources are neglected. In the hypothesis of isotropic mediums, the source term only depends on the scattering angle $\boldsymbol{\Omega}\cdot\boldsymbol{\Omega}'$. In this situation, it is common to expand the scattering cross section as a truncated series of Legendre polynomials. The particle balance is given by:

$$d^3\mathcal{A} = -d^3\mathcal{B} - d^3\mathcal{C} + d^3\mathcal{D} \qquad (2.19)$$

By removing the integrals due to the arbitrariness of the control volume C, we obtain the differential form of the transport equation:

$$\frac{1}{V_n}\frac{\partial}{\partial t}\phi(\boldsymbol{r},E,\boldsymbol{\Omega},t)+\boldsymbol{\Omega}\cdot\boldsymbol{\nabla}\phi(\boldsymbol{r},E,\boldsymbol{\Omega},t)+\Sigma(\boldsymbol{r},E)\phi(\boldsymbol{r},E,\boldsymbol{\Omega},t)=Q(\boldsymbol{r},E,\boldsymbol{\Omega},t) \tag{2.20}$$

2.2.2 Characteristic formulation

The characteristic form of the transport equation is obtained by considering the trajectory of the neutron over a characteristic, i.e. a straight line of direction $\boldsymbol{\Omega}$. In this way, by choosing a reference point $\boldsymbol{r}$ on its characteristic, the position of the neutron is given by $\boldsymbol{r}+s\boldsymbol{\Omega}$ at time $t+s/V_n$.

With this approach, the transport equation is rewritten in its characteristic form (here backward directed):

$$\begin{aligned}\frac{d}{ds}\phi(\boldsymbol{r}+s\boldsymbol{\Omega},E,\boldsymbol{\Omega},t+s/V_n)+\Sigma(\boldsymbol{r}+s\boldsymbol{\Omega},E)\phi(\boldsymbol{r}+s\boldsymbol{\Omega},E,\boldsymbol{\Omega},t+s/V_n)\\ =Q(\boldsymbol{r}+s\boldsymbol{\Omega},E,\boldsymbol{\Omega},t+s/V_n)\end{aligned} \tag{2.21}$$

Equation 2.21 can also be written in its forward form:

$$\begin{aligned}-\frac{d}{ds}\phi(\boldsymbol{r}-s\boldsymbol{\Omega},E,\boldsymbol{\Omega},t-s/V_n)+\Sigma(\boldsymbol{r}-s\boldsymbol{\Omega},E)\phi(\boldsymbol{r}-s\boldsymbol{\Omega},E,\boldsymbol{\Omega},t-s/V_n)\\ =Q(\boldsymbol{r}-s\boldsymbol{\Omega},E,\boldsymbol{\Omega},t-s/V_n)\end{aligned} \tag{2.22}$$

2.2.3 Integral formulation

The neutron flux at $\boldsymbol{r}$ can be expressed as the sum of all neutrons previously emitted by the source density Q and directed along $\boldsymbol{\Omega}$ that have followed a straight line trajectory without any interaction. This leads to the integral form of the transport equation., given by:

$$\phi(\boldsymbol{r},E,\boldsymbol{\Omega},t)=\int_0^\infty ds\ e^{-\tau(s,E)}Q(\boldsymbol{r}-s\boldsymbol{\Omega},E,\boldsymbol{\Omega},t-s/V_n) \tag{2.23}$$

This equation can also be obtained by integrating the angular flux along its characteristic and using equation 2.21.

The optical path $\tau(s,E)$ is defined by:

$$\tau(s,E)=\int_0^s ds'\ \Sigma(\boldsymbol{r}-s'\boldsymbol{\Omega},E) \tag{2.24}$$

The integrating factor $e^{-\tau(s,E)}$ in equation 2.23 represents the attenuation that would occur between the source point $\boldsymbol{r}-s\boldsymbol{\Omega}$ and the observation point $\boldsymbol{r}$. In particular, it represents the probability for a neutron born in $\boldsymbol{r} - s\boldsymbol{\Omega}$ with energy E and direction $\boldsymbol{\Omega}$ to travel up to $\boldsymbol{r}$.

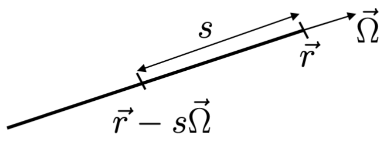


Figure 2.1 A neutron that travels a distance s on a characteristic of direction $\boldsymbol{\Omega}$

2.2.4 Steady state transport equation

Given the large time scale in which depletion effects manifest themselves, it is customary to simulate reactors under nominal conditions as if they were in steady-state conditions. In these conditions the reactor is said to be critical, i.e. there is a self-sustaining time-independent chain reaction in the absence of external sources of neutrons [3]. The time variable is then ignored, considering variations of the neutron flux with time to be slow enough to be neglected. In this way, equation 2.20 becomes:

$$\boldsymbol{\Omega} \cdot \boldsymbol{\nabla}\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}) + \Sigma(\boldsymbol{r}, E)\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}) = Q(\boldsymbol{r}, E, \boldsymbol{\Omega}) \tag{2.25}$$

An angular flux distribution that is solution of equation 2.25 is such that the rate of production of neutrons in any region of volume V is perfectly balanced by the sum of absorption and leakage rates. However, equation 2.25 with the source term defined as in 2.18, only has an elementary solution unless the volume composition and geometry are such that the reactor is critical. This condition is highly unlikely to be achieved numerically. [21].

In order to keep using equation 2.25 even for non critical systems, we introduce and adjustment parameter K_{eff} that divides the fission source term to maintain the balance for any reactor configuration. Accordingly, by neglecting external sources of neutrons and assuming an isotropic fission source, the steady state source density becomes:

$$\begin{aligned} Q(\boldsymbol{r}, E, \boldsymbol{\Omega}) = &\int_{4\pi} d^2\Omega' \int_0^\infty dE' \Sigma_s(\boldsymbol{r}, E \leftarrow E', \boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega}')\phi(\boldsymbol{r}, E', \boldsymbol{\Omega}') \\ &+ \frac{1}{4\pi K_{\text{eff}}} Q^{\text{fiss}}(\boldsymbol{r}, E) \end{aligned} \tag{2.26}$$

Or in the case of an expansion in Legendre polynomials:

$$Q(\boldsymbol{r}, E, \Omega) = \int_0^\infty dE' \sum_{l=0}^{L} \frac{2l+1}{4\pi} \Sigma_{s,l}(\boldsymbol{r}, E \leftarrow E') \sum_{m=-l}^{l} R_l^m(\boldsymbol{\Omega}) \phi_l^m(\boldsymbol{r}, E') + \frac{1}{4\pi K_{\text{eff}}} Q^{\text{fiss}}(\boldsymbol{r}, E) \tag{2.27}$$

Where $R_l^m(\boldsymbol{\Omega})$ functions are the real spherical harmonics, $\Sigma_{s,l}$ are the Legendre coefficients of the scattering cross section, while the spherical harmonics moments of the flux are expressed as:

$$\phi_l^m(\boldsymbol{r}, E) = \int_{4\pi} d^2\Omega \; R_l^m(\boldsymbol{\Omega}) \phi(\boldsymbol{r}, E, \boldsymbol{\Omega}) \tag{2.28}$$

The isotropic fission source is given by:

$$Q^{\text{fiss}}(\boldsymbol{r}, E) = \frac{1}{4\pi} \sum_{j=1}^{J} \chi_j(E) \int_0^\infty dE' \nu \Sigma_{f,j}(\boldsymbol{r}, E') \phi_0^0(\boldsymbol{r}, E') \tag{2.29}$$

The case of $K_{\text{eff}} > 1$ corresponds to a system in which neutron production rate is larger then neutron leakage and absorption rate. We say that this system is super-critical. On the other hand, $K_{\text{eff}} < 1$ corresponds to a system where neutron loss is faster then neutron production and hence the chain reaction is not sustained. The system is then sub-critical. Criticality is reached if $K_{\text{eff}} = 1$.

By substituting equation 2.26 into 2.25 we obtain an eigenproblem in which $1/K_{\text{eff}}$ is the eigenvalue and $\phi(\boldsymbol{r}, E, \boldsymbol{\Omega})$ is the eigenvector. Among the discrete eigenvalues of the problem, we call the fundamental solution the maximum possible value of K_{eff}. The fundamental solution, also known as effective multiplication factor, is also called the infinite multiplication factor when leakage rate is not considered. This is often the case in lattice calculations where we consider an infinite geometry. The angular flux corresponding to the fundamental solution is called fundamental mode and is the only solution that has a physical meaning, being the only solution that is positive over the whole geometry.

2.2.5 Boundary and continuity conditions

In order to solve the transport equation for a given volume V, the incoming flux at the boundaries of V has to be known *a priori*. Moreover, it is necessary to know how the flux behaves at the interface of different regions. For this reason we need to set up boundary and continuity conditions. The most common boundary conditions are presented below.

The vacuum boundary condition represents the situation in which there are no neutrons coming from the outside of the volume. This condition is mathematically represented by:

$$\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) = 0 \quad, \forall \boldsymbol{\Omega} : \boldsymbol{N} \cdot \boldsymbol{\Omega} < 0 \tag{2.30}$$

We remember that $\boldsymbol{N}$ is defined as the outward unit vector normal to the boundaries of V. Specular reflection boundary condition is instead given by:

$$\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) = \phi(\boldsymbol{r}, E, \boldsymbol{\Omega}', t) \quad, \boldsymbol{\Omega}' = \boldsymbol{\Omega} - 2(\boldsymbol{N} \cdot \boldsymbol{\Omega})\boldsymbol{N} \tag{2.31}$$

Both vacuum and specular reflection boundary conditions are a special case of the albedo boundary condition, defined as:

$$\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) = \beta\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}', t) \quad, \boldsymbol{\Omega}' = \boldsymbol{\Omega} - 2(\boldsymbol{N} \cdot \boldsymbol{\Omega})\boldsymbol{N} \tag{2.32}$$

In particular $\beta = 1$ corresponds to the specular reflection boundary condition while a null albedo corresponds to a vacuum boundary condition. Finally, the periodic boundary condition, commonly used in lattice calculations for assemblies or fuel pins, corresponds to the condition in which the volume V is repeated periodically in a lattice grid and hence the flux on one boundary will be equal to the flux on the opposite boundary:

$$\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t) = \phi(\boldsymbol{r} + \Delta\boldsymbol{r}, E, \boldsymbol{\Omega}, t) \tag{2.33}$$

Inside the volume V, the continuity condition requires that the angular flux $\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}, t)$ is continuous across all interfaces.

2.3 Solutions of the neutron transport equation

As seen in the previous section, the neutron transport equation is a complex integro-differential equation that cannot be solved analytically except for some simple cases. Therefore, numerical approaches are needed to obtain approximate solutions for realistic problems. Numerical methods can be classified into two main categories: stochastic methods and deterministic methods.

Stochastic methods, based on a Monte Carlo approach, do not solve the transport equation but rely on simulating the random behavior of individual neutrons and averaging the results over a large number of histories. Stochastic methods are very versatile and accurate, but are computationally expensive.

Deterministic methods rely on discretizing the neutron transport equation in space, energy, angle and time, and solving the resulting system of algebraic equations. Deterministic methods are usually faster than the stochastic ones, but they can also introduce numerical errors and approximations that affect the accuracy and validity of the solutions. For this reason, stochastic codes are generally used for providing benchmark calculations in order to validate deterministic schemes.

The forthcoming section is mainly focused on deterministic methods for solving the steady state transport equation. First, the energy variable is discretized by using the multigroup formalism. Then, angular and spatial variables are taken into account in different ways depending on the numerical technique considered.

The section is dedicated to introducing the main deterministic approaches utilized to solve the neutron transport equation. Initially, we shall briefly describe the fundamentals and applications of the spherical harmonics (P_N) and discrete ordinates (S_N) methods. These approaches find their applications also in full core calculations, since they offer a balance between computational efficiency and solution accuracy. The methods provide systematic solutions by either expanding the neutron flux in terms of spherical harmonics or discretizing the angular variable to a set of discrete ordinates, respectively.

However, the major emphasis of this discourse will be on the collision probability method (CP) and the method of characteristics (MOC), a specific formulation of the S_n method. Both CP and MOC occupy a cardinal position in the neutron transport computations, specially at the lattice level. The CP, which is renowned for its efficacy in one and two-dimensional reactor physics problems, essentially estimates the probability of neutron collision events.

Conversely, MOC uses a tracking procedure to model the neutron path, leading to a set of ordinary differential equations that are solved iteratively. This method is distinguished by its capacity for yielding directional fluxes, thereby standing as a preferred choice for detailed reactor physics analyses, specially when dealing with geometries containing materials with high anisotropic scattering cross sections.

2.3.1 Multigroup discretization

The deterministic approach for solving the transport equations consists in the discretization of the phase state variables $(\boldsymbol{r}, E, \boldsymbol{\Omega})$. All deterministic solvers use the so-called multigroup approach. This consists in discretizing the continous energy variable E into several energy intervals called groups. Accordingly, all neutronic properties such as angular flux, cross sections and source density must be averaged over the range of each group energies. This procedure is also known as energy condensation. For instance the group-averaged angular flux $\phi_g(\boldsymbol{r}, \boldsymbol{\Omega})$ will represent the integrated angular flux of all neutrons with energies in the group $E_g < E < E_{g-1}$.

The angular flux, source term and fission spectrum for group g appearing in the transport equation are defined as:

$$\phi_g(\boldsymbol{r}, \boldsymbol{\Omega}) = \int_{E_g}^{E_{g-1}} dE\ \phi(\boldsymbol{r}, E, \boldsymbol{\Omega}) \tag{2.34}$$

$$\phi_g(\boldsymbol{r}) = \int_{E_g}^{E_{g-1}} dE\ \phi(\boldsymbol{r}, E) \tag{2.35}$$

$$Q_g(\boldsymbol{r}, \boldsymbol{\Omega}) = \int_{E_g}^{E_{g-1}} dE\ Q(\boldsymbol{r}, E, \boldsymbol{\Omega}) \tag{2.36}$$

$$\chi_{j,g} = \int_{E_g}^{E_{g-1}} dE\ \chi_j(E) \tag{2.37}$$

Group-averaged cross sections are instead defined in such a way that the reaction rate is preserved:

$$\Sigma_g(\boldsymbol{r}) = \frac{1}{\phi_g(\boldsymbol{r})} \int_{E_g}^{E_{g-1}} dE\ \Sigma(\boldsymbol{r}, E)\phi(\boldsymbol{r}, E) \tag{2.38}$$

$$\Sigma_{s,g\leftarrow h}(\boldsymbol{r}) = \frac{1}{\phi_h(\boldsymbol{r})} \int_{E_g}^{E_{g-1}} dE \int_{E'_{h-1}}^{E'_h} dE'\ \Sigma_s(\boldsymbol{r}, E \leftarrow E')\phi(\boldsymbol{r}, E') \tag{2.39}$$

$$\nu\Sigma_{\mathrm{f,j,g}}(\boldsymbol{r}) = \frac{1}{\phi_g(\boldsymbol{r})} \int_{E_g}^{E_{g-1}} dE\ \nu\Sigma_{\mathrm{f,j}}(\boldsymbol{r}, E)\phi(\boldsymbol{r}, E) \tag{2.40}$$

It should be noted that the definition of group cross section constants requires knowledge of the neutron flux, which is generally unknown beforehand. In order to face this problem, the common approach is to replace the real flux by a reference flux obtained by a calculation with finer spectrum but simplified geometry. This approach will be detailed in 3.1 when discussing resonance self-shielding.

Using this formalism, the form of the transport equation for a group g remains the same in all its formulations. For instance, the multigroup differential form of the steady-state transport equation is given by:

$$\boldsymbol{\Omega} \cdot \boldsymbol{\nabla}\phi_g(\boldsymbol{r}, \boldsymbol{\Omega}) + \Sigma_g(\boldsymbol{r}, E)\phi_g(\boldsymbol{r}, \boldsymbol{\Omega}) = Q_g(\boldsymbol{r}, \boldsymbol{\Omega}) \tag{2.41}$$

Where the group-averaged source term is obtained by integration of equation 2.26:

$$Q_g(\boldsymbol{r}, \boldsymbol{\Omega}) = \sum_{h=1}^{G} \int_{4\pi} d^2\Omega' \Sigma_{s,g\leftarrow h}(\boldsymbol{r}, \boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega}')\phi_h(\boldsymbol{r}, \boldsymbol{\Omega}') \tag{2.42}$$

$$+ \frac{1}{4\pi K_{\text{eff}}} Q_g^{\text{fiss}}(\boldsymbol{r}) \tag{2.43}$$

with:

$$Q_g^{\text{fiss}}(\boldsymbol{r}) = \sum_{j=1}^{J} \chi_{j,g} \sum_{h=1}^{G} \nu\Sigma_{f,j,h}(\boldsymbol{r})\phi_h(\boldsymbol{r}, \boldsymbol{\Omega}) \tag{2.44}$$

As it will be foreseen in the next section, energy condensation is generally performed several times in the global computational scheme. A first condensation is performed starting from the continuous-energy cross sections libraries in order to produce a $50 < G < 400$ multigroup cross section library for lattice calculation. Then, depending on the lattice scheme, one or two more energy condensations are performed by the lattice code in order to retrieve a few-group ($2 \leq G \leq 8$) cross section library that will be used for core calculations. The number of groups used and the energy boundaries for each group must be chosen carefully since it may affect the precision and time spent for the simulation.

2.3.2 Angular approximations

Two commonly used methods for core calculation consist in discretizing the angular variable associated with neutron direction with a spherical harmonic expansion or a discrete ordinate method. The transport equation is solved for the given geometry using standard numerical analysis techniques such as finite difference or finite element approach.

Discussion will be limited only to the angular discretization techniques. More details regarding the finite difference approach and finite element method can be found respectively in references [3] and [22].

Spherical harmonics method

The spherical harmonics method, or P_n method, consists in approximating the angular dependency of the flux by expanding the angular flux in a truncated series of spherical harmonics polynomials. Among the advantages of the P_n method is the fact that the expansion coefficients for the angular flux are the same as those appearing in the source term of the transport equation due to the expansion of the scattering kernel [3].

$$\phi(\boldsymbol{r}, \boldsymbol{\Omega}) \approx \sum_{l=0}^{n} \frac{2l+1}{4\pi} \sum_{m=-l}^{l} \phi_l^m(\boldsymbol{r}) R_l^m(\boldsymbol{\Omega}) \tag{2.45}$$

with:

$$\phi_l^m(\boldsymbol{r}) = \int_{4\pi} d^2\Omega \; \phi(\boldsymbol{r}, \boldsymbol{\Omega}) R_l^m(\boldsymbol{\Omega}) \tag{2.46}$$

An expansion up to the n-th term lead to a set of $n+1$ coupled equations (called P_n equations) for each energy group that need to be discretized and solved. For two or three dimensional geometry this leads to a linear system that takes too much computational effort to be solved.

For this reason, a simplified P_n method (SP_n) is generally used. An heuristic derivation of these simplified equations start from the P_n equations applied to a 1D slab geometry:

$$\frac{l}{2l+1} \frac{d}{dx} \phi_{l-1}(x) + \frac{l+1}{2l+1} \frac{d}{dx} \phi_{l+1}(x) + \Sigma(x)\phi_l(x) = Q_l(x) \tag{2.47}$$

where ϕ_l and Q_l are the l-th Legendre moments of the flux and source density respectively. Then, depending on the value of l, the derivative $\frac{d}{dx}$ is replaced by:

1. The divergence operator for even l
2. The gradient operator for odd l

Accordingly, the SP_n equations are given by:

$$\frac{l}{2l+1} \boldsymbol{\nabla} \cdot \boldsymbol{\phi}_{l-1}(\boldsymbol{r}) + \frac{l+1}{2l+1} \boldsymbol{\nabla} \cdot \boldsymbol{\phi}_{l+1}(\boldsymbol{r}) + \Sigma(\boldsymbol{r})\phi_l(\boldsymbol{r}) = Q_l(\boldsymbol{r}) \tag{2.48}$$

for $0 \leq l \leq n-1$ and even l, and:

$$\frac{l}{2l+1} \boldsymbol{\nabla} \phi_{l-1}(\boldsymbol{r}) + \frac{l+1}{2l+1} \boldsymbol{\nabla} \phi_{l+1}(\boldsymbol{r}) + \Sigma(\boldsymbol{r})\boldsymbol{\phi}_l(\boldsymbol{r}) = \boldsymbol{Q}_l(\boldsymbol{r}) \tag{2.49}$$

with $1 \leq l \leq n$ and odd l.

Discrete ordinate method

The discrete ordinate approximation, or S_n approximation, consists of requiring equation 2.25 to hold only for some discrete number of directions $\boldsymbol{\Omega}_n$, and to apply a compatible quadrature to approximate the integral terms. In this way, the equation to be solved is:

$$\boldsymbol{\Omega}_n \cdot \boldsymbol{\nabla}\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}_n) + \Sigma(\boldsymbol{r}, E)\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}_n) = Q(\boldsymbol{r}, E, \boldsymbol{\Omega}_n) \tag{2.50}$$

By solving equation 2.50 for each direction we can determine the neutron flux by mean of a quadrature rule. The most commonly used rule is the Level Symmetric (LS) quadrature, which applies to systems in which the angular dependency satisfies reflective conditions across the x, y and z plane in such a way that only one octant of the sphere needs to be considerered when integrating over all possible directions. LS quadratures use the same set of $N/2$ positive values of the direction cosines with respect to each of the three axes. With this approach, ordinates are arranged on the unit octant as shown in Figure 2.2.

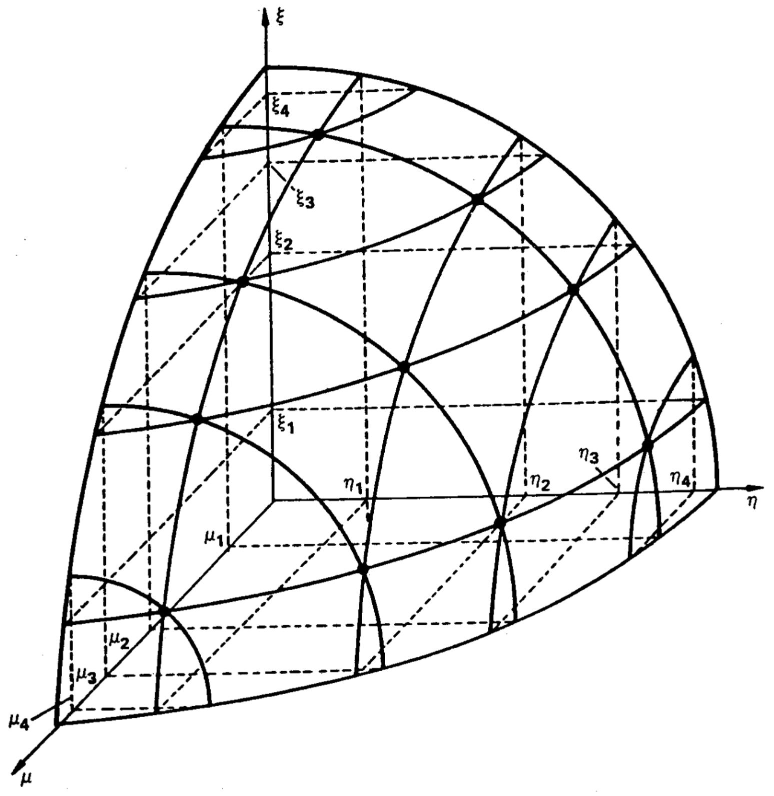


Figure 2.2 LS S_8 discrete ordinates set [3]

In this case the neutron flux can be approximated by :

$$\phi(\boldsymbol{r}, E) = \int_{4\pi} d^2\Omega \phi(\boldsymbol{r}, E, \boldsymbol{\Omega}) \approx \sum_{n=1}^{N(N+2)} w_n \phi(\boldsymbol{r}, E, \boldsymbol{\Omega}_n) \tag{2.51}$$

with $N(N+2)/8$ represents the number of discrete ordinates per octant and w_n is the weight [3].

2.3.3 Collision probability method

The interpretation of the integral form of the transport equation is that the neutron flux $\phi_g(\boldsymbol{r}, \boldsymbol{\Omega})$ is given by the sum of all neutrons previously emitted by a source density Q, directed along $\boldsymbol{\Omega}$, and that have followed a straight line trajectory directed towards $\boldsymbol{r}$ without any interaction. This equation, in its steady-state, multigroup form is given by:

$$\phi_g(\boldsymbol{r}, \boldsymbol{\Omega}) = \int_0^\infty ds\ e^{-\tau_g(s)} Q_g(\boldsymbol{r} - s\Omega, \Omega) \tag{2.52}$$

Where the optical path in group g is given by:

$$\tau_g(s) = \int_0^s ds'\ \Sigma_g(\boldsymbol{r} - s'\boldsymbol{\Omega}) \tag{2.53}$$

The collision probability (CP) method is the result of the spatial discretization of 2.52 assuming an isotropic source density. In lattice calculations, the transport equation is generally solved for an infinite lattice geometry, made out of identical cells, repeating themselves by symmetric periodic boundary conditions. Accordingly, the lattice geometry can be seen as a reference unit cell, surrounded by an infinite sequence of phantom cells, identical to the reference one. The reference unit cell is composed by different homogeneous regions V_i.

With such a geometry, the collision probability method states that the averaged integrated neutron flux in region j and group g is such that:

$$V_j \Sigma_{j,g} \phi_{j,g} = \sum_i Q_{i,g} V_i P_{ij,g} \tag{2.54}$$

with:

$$\Sigma_{j,g} = \frac{1}{V_j \phi_{j,g}} \int_{V_j} d^3r\ \Sigma_g(\boldsymbol{r}) \phi_g(\boldsymbol{r}) \tag{2.55}$$

$$P_{ij,g} = \frac{1}{4\pi V_i} \int_{V_i^\infty} d^3r' \int_{V_j} d^3r\ \Sigma_g(\boldsymbol{r}) \frac{e^{-\tau_g(s)}}{s^2} \tag{2.56}$$

The collision probability $P_{ij,g}$ is the probability for a neutron born in any of the regions V_i (i.e. from the reference or phantom cells, noted V_i^∞) to undergo its first collision in region V_j^∞. Equation 2.54 states that the reaction rate of neutrons in region j is given by the sum over every region V_i of the production rate of neutrons weighted by their corresponding collision probability.

It is of common use to refer to the reduced collision probabilities, defined as:

$$p_{ij,g} = \frac{P_{ij,g}}{\Sigma_{j,g}} \tag{2.57}$$

By using the collision probability properties of reciprocity and conservation, the averaged integrated angular flux is written as:

$$\phi_{i,g} = \sum_j Q_{i,g} p_{ij,g} \tag{2.58}$$

Note that the source term appearing in equation 2.58 is dependent on the flux, which means that an iterative process is required in order to find a good approximation for $\phi_{i,g}$.

Numerical implementation of the CP method

In order to implement a collision probability solver the first step is to compute the collision probability matrix $P_{ij,g}$ for each energy group, then the integrated flux can be determined by solving equation 2.58.

Collision probability matrix is generally determined in the following way:

- From the problem geometry, a discrete number of integration lines are traced by the mean of a tracking procedure. This procedure consists in computing the intersections points between the integration lines and the surfacic elements of the geometry in order to retrieve some weights coefficents allowing to compute the spatial integrals by the mean of a simple quadrature formula.
- A collision probability matrix for each energy group is then computed numerically by using the integration weights derived from the tracking procedure. This process can be done in parallel since there is no interaction between different energy groups in equation 2.56.

Equation 2.58 is instead solved by using an iterative process. The first step is to rewrite equation 2.58 in the form of:

$$\boldsymbol{\phi}_g = \mathbb{W}_g \boldsymbol{Q}_g^{\diamond} \tag{2.59}$$

where $\boldsymbol{\phi}_g = \{\phi_{i,g}, \forall i\}$ and $\boldsymbol{Q}_g^{\diamond} = \{Q_{g,i}^{\diamond}, \forall i\}$ while $\mathbb{W}_g$ is the so-called scattering-reduced collision probability matrix. The term $Q_{g,i}^{\diamond}$ is the fission and scattering source term of group g and region i without the intra-group diffusion term. The scattering reduced matrix and the $Q^{\diamond}_{g,i}$ are defined as:

$$\mathbb{W}_g = [\mathbb{I} - \mathbb{P}_g \mathbb{S}_{s0,g\leftarrow g}]^{-1}\mathbb{P}_g \tag{2.60}$$

$$Q^{\diamond}_{g,i} = \sum_{h\neq g} \Sigma_{s0,i,g\leftarrow h}\phi_{i,h} + \frac{1}{K_{\text{eff}}} Q^{fiss}_{i,g} \tag{2.61}$$

where $\mathbb{I}$ is the identity matrix, $\mathbb{P}_g$ is the collision probability matrix of group g and $\mathbb{S}_{s0,g\leftarrow g}$ is a diagonal matrix made of $\Sigma_{s0,i,g\leftarrow g}$. The resulting source term in equation 2.59 is due to the sum of two contributions having opposite effects on the neutron flux. On one hand, the diffusion term has the tendency of slowing down neutrons and hence bringing neutrons coming from fast groups towards thermal groups. On the other hand, fission source is due to thermal neutrons producing fast neutrons.

For this reason, equation 2.59 is solved using a double iteration scheme. The outer iteration (also known as power iteration) is performed over the fission source term by changing the value of the effective multiplication factor K_{eff} at each iteration. The inner iteration is performed by iterating over the diffusion source term by keeping a constant value of the fission source term. In this way convergence of both K_{eff} and flux is ensured.

Conclusions on the CP method

A first limitation of the method is the assumption of isotropic source, which reduces the accuracy when considering a medium having anisotropic scattering. The major limitation of the CP method derives from the fact that for a geometry composed by I regions, the method will produce, for each energy group, a $I \times I$ matrix that will be used to solve an algebraic system. This means that while this method is highly recommended for few regions geometries, geometries made out of several regions can cause memory problems, due to the quadratic dependance of the system matrix, or lead to long computational times.

This is the reason why this method is currently used in industrial schemes only for self-shielding calculations, where geometries are simplified with respect to the ones for flux calculation.

This limitation is partially solved by the interface current (IC) method, which divides the main geometry in smaller regions, applies the CP method for each of these regions, then reconstructs the flux from the knowledge of the interface currents surrounding each region.

2.3.4 Method of Characteristics

The Method of Characteristics (MOC) addresses the characteristic form of the transport equation by tracing the straight paths of neutrons as they move through the domain. This method relies on an iterative process to calculate the particle flux by solving the transport equation along tracks that span the entire domain. The tracking procedure is the same as the one used for the CP method. The following presentation of the method is inspired from reference [23].

Tracking procedure

Let us consider a geometry made out of several homogeneous regions V_i. A tracking procedure allows to generate a set of integration lines (characteristics) over the geometry that physically represent the trajectories of neutrons when crossing the reference domain. A trajectory $\boldsymbol{T}$ is defined by an orientation $\boldsymbol{\Omega}$ and a starting point $\boldsymbol{p}$ that belongs to a plane $\pi_{\boldsymbol{\Omega}}$ perpendicular to $\boldsymbol{\Omega}$, as seen in Figure 2.3.

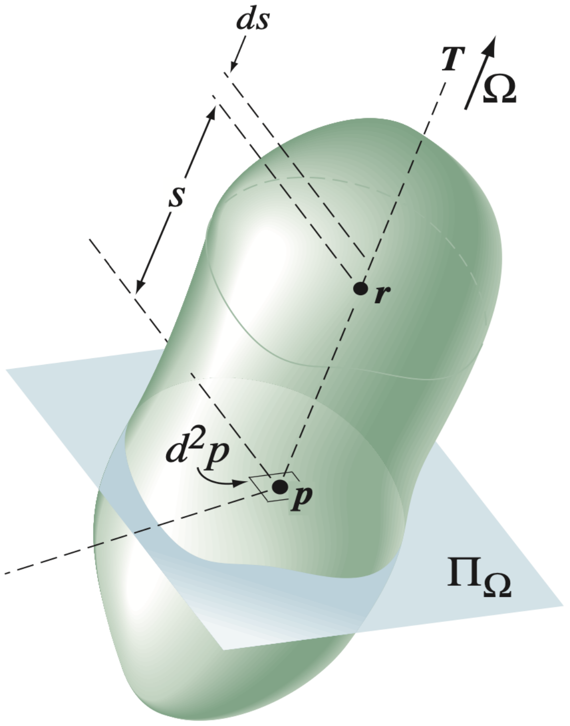


Figure 2.3 Tracking formalism [4]

Integration lines must be traversed back and forth since neutrons can travel in both $\boldsymbol{\Omega}$ and $-\boldsymbol{\Omega}$ directions.

For a given integration line $\boldsymbol{T}(\boldsymbol{\Omega}, \boldsymbol{p})$, the data required for a numerical integration are the indices $N_k(\boldsymbol{T})$ of the crossed regions and the tracking lengths $L_k(\boldsymbol{T})$, which represent the lengths of the intersections between $\boldsymbol{T}$ and the region V_k. With this notation the intersections $\boldsymbol{r}_k$ between the integration line and the boundaries of each region are given by:

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k + L_k\boldsymbol{\Omega} \quad , k \in [1, K] \tag{2.62}$$

Note that $\boldsymbol{r}_1$ is the entry point of the integration line on the domain, while $\boldsymbol{r}_{K+1}$ represents the exit point. The averaged flux over each segment is given by:

$$\bar{\phi}_k = \frac{1}{L_k}\int_0^{L_k} ds\ \phi(\boldsymbol{r}_k + s\boldsymbol{\Omega}, \boldsymbol{\Omega}) \tag{2.63}$$

Integration over tracked regions

Using the tracking procedure, we have the following identity for a given function $f(\boldsymbol{r}, \boldsymbol{\Omega})$:

$$\int_{V_i} d^3r \int_{4\pi} d^2\Omega f(\boldsymbol{r}, \boldsymbol{\Omega}) = \int_{\mathcal{T}} d^4T\ \sum_k \delta_{i,N_k} \int_{s_k}^{s_{k+1}} ds\ f(\boldsymbol{p} + s\boldsymbol{\Omega}, \Omega) \tag{2.64}$$

Where $\mathcal{T}$ is the set of all tracking lines, and δ_{i,N_k} is the Kronecker function. In this way, volumes and surfaces for each region can be expressed as functions of the tracking information:

$$\tilde{V}_i = \frac{1}{4\pi}\int_{\mathcal{T}} d^4T \sum_k \delta_{i,N_k} L_k(\boldsymbol{T}) \tag{2.65}$$

$$\tilde{S}_\alpha = \frac{1}{\pi}\int_{\mathcal{T}} d^4T\ \chi_\alpha(\boldsymbol{T}, \boldsymbol{r}) \tag{2.66}$$

Where $\chi_\alpha(\boldsymbol{T}, \boldsymbol{r})$ is equal to 1 if point $\boldsymbol{r}$ on $\boldsymbol{T}$ crosses surface α, and zero otherwise. Tracking procedure retrieves weights that can be used to numerically compute the integrals:

$$\int_{\mathcal{T}} d^4T\ f(\boldsymbol{T}) \approx 4\pi \sum_{m=1}^{M} \omega_m f(\boldsymbol{T}_m) \tag{2.67}$$

Where M is the number of discrete characteristics. Volumes and surfaces are generally computed numerically and can be used for renormalisation purposes. For example, numerical volumes can be used to renormalize the lengths of each integration line:

$$L_k(\boldsymbol{\Omega}, \boldsymbol{p}) \leftarrow L_k(\boldsymbol{\Omega}, \boldsymbol{p})\frac{V_j}{\tilde{V}_j(\boldsymbol{\Omega})} \tag{2.68}$$

Solution procedure

The flux is computed by solving the characteristic form of the transport equation 2.21. For a given integration line $T_m(\boldsymbol{\Omega}_m, \boldsymbol{p}_m)$, directed along $\boldsymbol{\Omega}$, we then need to solve sequentially the following equation:

$$\frac{d\phi}{ds}(\boldsymbol{r}_k + s\boldsymbol{\Omega}, \boldsymbol{\Omega}) + \Sigma_{t,N_k}\phi(\boldsymbol{r}_k + s\boldsymbol{\Omega}, \boldsymbol{\Omega}) = Q_k(\boldsymbol{r}_k + s\boldsymbol{\Omega}, \boldsymbol{\Omega}) \quad , s \in [0, L_k] \tag{2.69}$$

Here Σ_{t,N_k} is the macroscopic cross section of the region having index N_k. In order to solve equation 2.69, knowledge on the variation of the source term is required. The common approach is to use a flat-source approximation, consisting in assuming a constant source term in each region for a given direction, so that: $Q_k(\boldsymbol{r}_k + s\boldsymbol{\Omega}, \boldsymbol{\Omega}) = Q_k(\boldsymbol{\Omega}), \forall s \in [0, L_k]$. With this approximation, equation 2.69 can be analytically integrated over the segment L_k, leading to the following incremental scheme, also known as the step characteristic scheme:

$$\phi_{k+1}(\boldsymbol{T}) = \phi_k(\boldsymbol{T})e^{-\tau_k} + Q_{N_k}(\boldsymbol{\Omega})\frac{1 - e^{\tau_k}}{\Sigma_{t,N_k}} \tag{2.70}$$

Where $\phi_{k+1}(\boldsymbol{T}) = \phi(\boldsymbol{r}_k, \boldsymbol{\Omega})$ and $\tau_k = L_k\Sigma_{t,N_k}$. A similar equation is obtained for the averaged flux:

$$L_k\bar{\phi}_k(\boldsymbol{T}) = \phi_k(\boldsymbol{T})\frac{1 - e^{\tau_k}}{\Sigma_{t,N_k}} + Q_{N_k}(\boldsymbol{\Omega})\frac{L_k}{\Sigma_{N_k}}\left(1 - \frac{1 - e^{-\tau_k}}{\tau_k(\boldsymbol{T})}\right) \tag{2.71}$$

The scheme can be solved starting from the incoming angular flux ϕ_1 for each tracking line. The knowledge of this flux depends on the boundary conditions and on the type of tracking that is applied.

The iterative scheme is similar to the one presented for the collision probability method. Starting from a guess value for the source term, the angular flux can be computed along every integration line, allowing to determine a new value for the source term. An inner iteration over the diffusion term ensures the convergence of the angular flux, while the outer iteration, over the fission term, lead to a converged value of the effective multiplication factor K_{eff}. In order to reduce computational time, adequate initialization and acceleration techniques of the flux are essential.

Conclusions on the MOC approach

This method overcomes two main limitations that are found in the collision probability method. Not only the method takes into account scattering anisotropy, but is also a fully iterative scheme. This implies that the method can be used to solve the transport equation over complex geometries made out of several thousands of regions. For this reason, the MOC is used in several industrial codes for calculations where the geometries are finely refined.

CHAPTER 3 Elements of lattice calulation

Nuclear reactors' design, deployment, and operation heavily rely on validated numerical deterministic tools. Currently, neutronic studies are executed in two steps: lattice and core calculations. Lattice calculations solve the multigroup transport equation over a reduced geometry portion (cell or assembly) typically treated with 2D geometries and provide the required data to run a 3D core calculation. This chapter describes the fundamental aspects that underpin deterministic lattice codes and their essential components.

In thermal reactors, fast neutrons are slowed down by successive collisions with moderating materials before inducing a fission reaction. During this process, their energy varies from few MeV to fractions of eV, passing through the epithermal energy regions. Heavy nuclide cross sections exhibit strong variations called resonances in this energy range. Localized depressions of the neutron flux in energy and space, caused by resonant absorptions, are addressed with the help of self-shielding modules. These modules modify the nuclear cross-sections of resonant isotopes, allowing for a more accurate determination of the neutron flux.

Flux calculations in lattice codes typically assume reflection, translation, and rotation boundary conditions, which do not allow for a direct calculation of the leakage rates from the reference cell. Leakage models are used to estimate leakage rates from the cell or assembly.

Once the neutron flux has been computed, lattice codes average nuclear properties defined on heterogeneous assemblies into homogeneous mediums, producing nuclear data for core calculations.

Finally, a reactor database is produced, storing nuclear properties of the geometry under consideration at different operating conditions such as burnup, boron concentration, and coolant temperature. These reactor databases, stored in a multi-parameter library (MPO, MULTICOMPO, APEX, or SAPHYB, depending on the code) files, are used as input for full-core calculations.

3.1 Self-shielding calculations

The multigroup approach resulted in defining flux-dependent group-averaged quantities. The flux energy and spatial distribution rapidly change at the resonance energies of heavy isotopes. In such instances, a flux depression occurs, exhibiting behavior opposite to the resonances. Therefore, when a material has a high macroscopic cross-section for neutron absorption, it can significantly shield itself from neutron penetration, meaning that the inner portions of the material experience a reduced neutron flux compared to the outer regions.

This effect is particularly important for fuel pins, where the rim effect is present. The thermal flux is mostly absorbed in the regions close to the outer pin surface; therefore, these regions will have a much higher reaction rate than the more internal ones [4].

To properly define the macroscopic cross sections so that they can preserve the reaction rate, it is initially necessary to carry out a calculation on a simplified geometry but with a finer energy spectrum that allows obtaining an estimate of the flux (called resonant flux) that mirrors the essential spatial variations observed. Then, spatially-dependent group-averaged cross sections that account for the self-shielding effect are computed. Particular attention must be paid to the fuel pins, which must be divided into several concentric rings, as depicted in Figure 3.1 as they are the most subjected to self-shielding effects and have the greatest effect on the K_{eff} and on the flux distribution. The number of rings is chosen depending on the purpose of the calculation scheme. An industrial scheme needs to be fast; hence, fewer rings are used while more are found in reference schemes. Figure 4.2a shows typical geometries for self-shielding calculations.

Considering reflectors, self-shielding is of particular importance when dealing with heavy reflectors. These are mainly made out of steel and present high-energy resonances due to the presence of ^{56}Fe (see Figure 3.2). In this situation, reflector self-shielding is generally performed on a one-dimensional or thin slab.

Self-shielding calculations rely on the resolution of a simplified form of the transport equation, determined after applying the Livolant-Jeanpierre approximations [24]. Two primary approaches are generally used to address this. The first, the sub-group method, solves a simplified form of the transport equation by leveraging almost continuous cross-section libraries, or "autolibs", which contain tens to hundreds of thousands of energy groups for resonant isotopes. The second approach, centered on equivalence in dilution, computes the resonant flux in a homogenous media equivalent to the heterogenous one. Despite the substantial computational effort these techniques require, they remain fundamental in achieving precise and reliable outcomes in lattice calculations.

3.1.1 Livolant-Jeanpierre approximations

Solving the transport equation, even for simplified geometries, using the Boltzmann equation as it is with autolib cross sections, is way too computationally expensive. For this reason, Livolant and Jeanpierre derived some approximations that lead to a simplified form of the

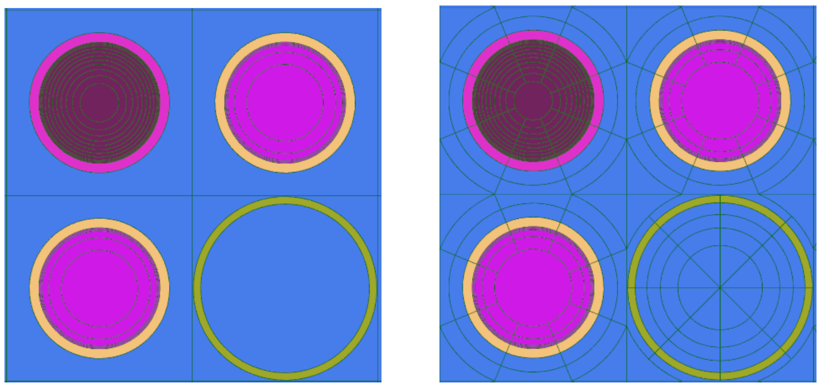

(a) Self-shielding geometry (b) Flux geometry

Figure 3.1 Typical meshes used for self-shielding and flux calculation. The top-left pin cell is a fuel assembly with Gadolinium; the bottom-right is a guide tube, while the remaining cells are standard fuel pins.

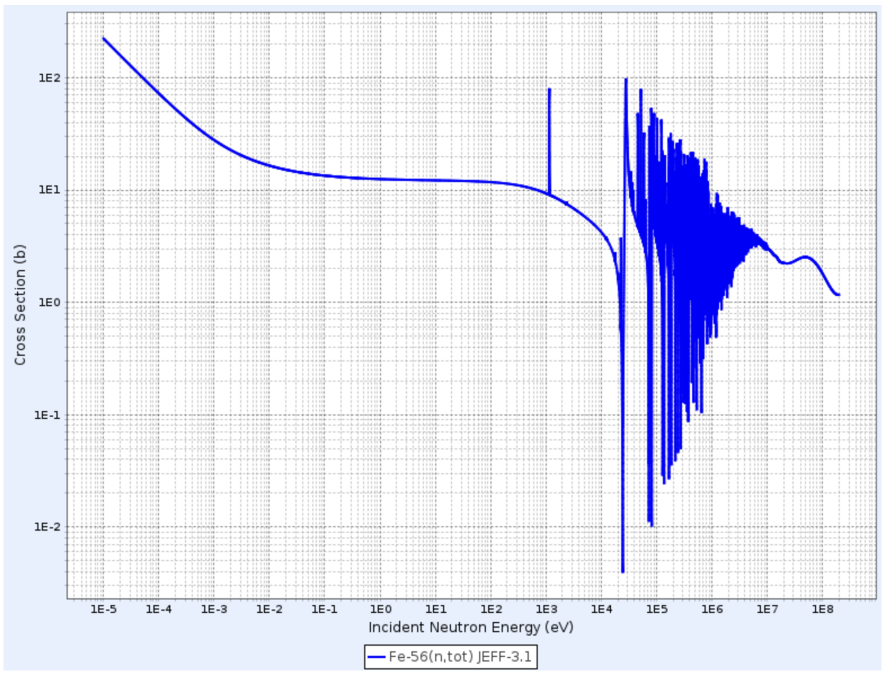


Figure 3.2 ^{56}Fe total cross sections for neutrons [5]

transport equation, allowing to perform self-shielding calculations with reduced computational time [24].

Starting from the Boltzmann equation with isotropic and elastic scattering, the first approximation assumes that thermal effects and sources originating from inelastic scattering, fission, or (n, xn) reactions are negligible in the slowing-down energy domain. Accordingly, the Boltzmann equation can be expressed as:

$$\boldsymbol{\Omega} \cdot \boldsymbol{\nabla}\phi(\boldsymbol{r}, u, \boldsymbol{\Omega}) + \Sigma(\boldsymbol{r}, u)\phi(\boldsymbol{r}, u, \boldsymbol{\Omega}) = \frac{1}{4\pi}[\mathcal{R}^+(\phi(\boldsymbol{r}, u)) + \mathcal{R}^*(\phi(\boldsymbol{r}, u))] \tag{3.1}$$

u is the lethargy defined as $u = \ln(E_0/E)$, where E_0 is a reference energy typically larger than 10 MeV. $\mathcal{R}^+(\phi(\boldsymbol{r}, u))$ and $\mathcal{R}^*(\phi(\boldsymbol{r}, u))$ are the slowing-down operators of nonresonant and resonant isotopes respectively. These operators are expressed as functions of the zeroth Legendre moments of the differential scattering cross sections of nonresonant and resonant isotopes:

$$\mathcal{R}^+(\phi(\boldsymbol{r}, u)) = \int_0^\infty du' \; \Sigma_{s0}^+(\boldsymbol{r}, u \leftarrow u')\phi(\boldsymbol{r}, u) \tag{3.2}$$

$$\mathcal{R}^*(\phi(\boldsymbol{r}, u)) = \int_0^\infty du' \; \Sigma_{s0}^*(\boldsymbol{r}, u \leftarrow E')\phi(\boldsymbol{r}, u) \tag{3.3}$$

It is also assumed that the domain under consideration contains only a single resonant isotope. Accordingly, self-shielding is performed one isotope at a time. Then, the next assumption considers the neutron flux as the product of a resonant fine-structure function $\varphi(\boldsymbol{r}, u)$, representing the rapid depression of the flux at resonant energies, and a distribution $\psi(\boldsymbol{r}, u)$. This latter function is assumed to be equal to:

$$\psi(\boldsymbol{r}, u) = \frac{\mathcal{R}^+(\phi(\boldsymbol{r}, u))}{\Sigma_s^+(\boldsymbol{r}, u)} \tag{3.4}$$

Since resonant isotopes are typically heavy, the maximum lethargy gain after a scattering reaction is small. Accordingly $\psi(\boldsymbol{r}, u)$ will change very slightly [25], resulting in :

$$\mathcal{R}^*(\phi(\boldsymbol{r}, u)) = \psi(\boldsymbol{r}, u)\mathcal{R}^*(\varphi(\boldsymbol{r}, u)) \tag{3.5}$$

Finally, it is assumed that $\psi(\boldsymbol{r}, u)$ is constant: $\psi(\boldsymbol{r}, u) = \psi(u)$. These approximations lead to a simplified form of the transport equation that depends only on the resonant fine-structure function:

$$\boldsymbol{\Omega} \cdot \boldsymbol{\nabla}\varphi(\boldsymbol{r}, u, \boldsymbol{\Omega}) + \Sigma(\boldsymbol{r}, u)\varphi(\boldsymbol{r}, u, \boldsymbol{\Omega}) = \frac{1}{4\pi}[\Sigma_s^+(\boldsymbol{r}, u) + \mathcal{R}^*(\varphi(\boldsymbol{r}, u))] \tag{3.6}$$

This equation is solved using a collision probability approach in self-shielding modules of lattice codes. This equation is the basis of all the self-shielding methods that are used.

3.1.2 Sub-group method

The sub-group method transforms the resonant cross-section structure in the autolib into probability tables and replaces the Riemann integrals appearing in the slowing-down operator $\mathcal{R}^*$ into Lebesgue ones. That is:

$$\frac{1}{x_2 - x_1} \int_{x_1}^{x_2} dx \; g[\sigma(x)] = \int_{\min \sigma}^{\max \sigma} d\sigma \; \pi(\sigma) g(\sigma) \tag{3.7}$$

with:

$$\int_{\min \sigma}^{\max \sigma} d\sigma \; \pi(\sigma) = 1 \tag{3.8}$$

The support of the probability density $\pi(\sigma)$ is $\{\Omega : [\min \sigma, \max \sigma]\}$. Probability tables provide the coefficients ω_k to calculate Lebesgue integrals using quadrature formulas. $\pi(\sigma)$ is approximated by an equivalent probability density having support Ω, and expressed as a sum of Dirac distributions [4]:

$$\Pi(\sigma) = \sum_k \delta(\sigma - \sigma_k)\omega_k \tag{3.9}$$

with:

$$\sum_k \omega_k = 1 \tag{3.10}$$

Starting from the Livolant-Jeanpierre constitutive equation, the approach is to define for each group g, a number K_g of subgroups. The number of sub-groups depends on the number of resonances in group g. The resulting sub-group equation is generally solved using a collision probability approach to determine the flux for each region and sub-group. The self-shielded cross section for reaction x in region i and group g is then given by [4]:

$$\sigma_{x,i,g} = \frac{\int_{u_g}^{u_{g-1}} du \; \sigma_{x,i}(u)\phi_i(u)}{\int_{u_g}^{u_{g-1}} du \; \phi_i(u)} = \frac{\sum_{k_g=1}^{K} \omega_{k_g} \sigma_{x,i,k_g} \varphi_{i,k_g}}{\sum_{k_g=1}^{K} \omega_{k_g} \varphi_{i,k_g}} \tag{3.11}$$

3.1.3 Equivalence in dilution models

In the case of an infinite homogeneous medium, equation 3.6 can be simplified in the following form, called the flux calculator formula:

$$(\sigma_e + \sigma^*(u))\varphi(u) = \gamma\sigma_e + \frac{\mathcal{R}^*(\phi(\boldsymbol{r}, u))}{N^*} \tag{3.12}$$

Where N^* is the number density of the resonant isotope, $\sigma^*(u)$ is the microscopic total cross-section of the resonant isotope. σ_e and γ are called dilution and gamma factor respectively, and are expressed as:

$$\sigma_e = \frac{\Sigma^+}{N^*} \quad \text{and} \quad \gamma = \frac{\Sigma_s^+}{\Sigma^+} \tag{3.13}$$

The flux calculator formula is then solved rapidly, being a zero-dimensional equation. Values of cross-sections are then condensed and tabulated as a function of temperature and dilution. Codes like NJOY carry out this procedure [26], and the resulting data is used as an input for the lattice calculation.

Expressions for the resonant fine structure flux are available for a homogeneous media considering some approximations. When resonances are treated as isolated and narrow, the narrow resonance model is applicable, leading to:

$$\varphi_{NR}(u) = \gamma \frac{\sigma_p^* + \sigma_e}{\sigma_e + \sigma^*(u)} \tag{3.14}$$

If the resonances are large, then the wide resonance model is applied and:

$$\varphi_{WR}(u) = \gamma \frac{\sigma_e}{\sigma_e + \sigma^*(u) - \sigma_s^*(u)} \tag{3.15}$$

Equivalence in dilution models consists of determining, for each resonant isotope, the corresponding equivalent dilution to be used to determine the self-shielded cross-sections by interpolating the tabulated data. Different methods are available, some are based on rational expansions of fuel-to-fuel collision probabilities [27, 28]. APOLLO2 and APOLLO3® use instead the Sanchez-Coste or *fine structure method* (FSM), an equivalence in dilution model presented in Ref. [29]. The FSM uses probability tables to solve both the heterogeneous and homogeneous cases.

3.2 Leakage calculations

Most lattice calculations consider a geometry where a reference 2D cell is repetitively arranged throughout the space, consequently forming a lattice. Typically, the conditions imposed on the cell pertain to reflection or translation, which stems from the foundational objective of determining the flux and reaction rates of a cell or assembly without acknowledging the surrounding components.

However, this approach does not inherently determine the leakage rate at the cell boundaries. As such, it becomes imperative to apply a leakage model to acquire an estimation of the leakage rate. Notably, these models also enable the derivation of diffusion coefficients to be later utilized, if required, in core calculations.

The neutron leakages are thus adjusted in each group g, assuming normal operating conditions, ensuring the cell remains in steady-state conditions (i.e., $K_{\text{eff}} = 1$). This is typically achieved through the introduction of the fundamental mode approximation, which assumes that the flux can be expressed as:

$$\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}) = \psi(\boldsymbol{r})\varphi(\boldsymbol{r}, E, \boldsymbol{\Omega}) \tag{3.16}$$

Here, $\varphi(\boldsymbol{r}, E, \boldsymbol{\Omega})$ denotes the homogeneous or periodic fundamental flux, representing the local flux variations in the cell due, for instance, to changes in material composition. Meanwhile, $\psi(\boldsymbol{r})$ illustrates the global flux curvature, independent of energy and neutron direction. The function $\psi(\boldsymbol{r})$ is assumed to satisfy the Laplace equation:

$$\nabla^2\psi(\boldsymbol{r}) + B^2\psi(\boldsymbol{r}) = 0 \tag{3.17}$$

where the buckling B^2 is the parameter adjusted to dictate the curvature of $\psi(\boldsymbol{r})$ to render the system critical. The form of the function $\psi(\boldsymbol{r})$ is given by the general solution:

$$\psi(\boldsymbol{r}) = \psi_0 e^{i\boldsymbol{B}\cdot\boldsymbol{r}} \tag{3.18}$$

This leads to the flux assuming the following form:

$$\phi(\boldsymbol{r}, E, \boldsymbol{\Omega}) = \varphi(\boldsymbol{r}, E, \boldsymbol{\Omega}) e^{i\boldsymbol{B}\cdot\boldsymbol{r}} \tag{3.19}$$

Leakage rates and diffusion coefficients are calculated based on the flux shape detailed in equation 3.19 and utilized in solving the transport equation. Homogeneous models, like the

B_1 leakage model, operate by computing the leakage rates over a fully homogenized cell, consequently solving the transport equation within a homogeneous medium. Conversely, heterogeneous models employ a heuristic strategy that integrates the homogeneous models with the heterogeneous effects derived from a CP calculation.

The current study uses a lattice code to carry out full-core reference calculations on a 2D geometry. Under these circumstances, axial leakage is neglected.

3.3 Homogenization techniques

As previously mentioned, simulations of reactor cores using transport are very time-consuming, and thus, they are rarely used for industrial applications. Instead, cores are more commonly simulated using the diffusion approximation. However, this approximation works best on homogeneous media, unlike the heterogeneous media that constitute most reactors. Generally, lattice codes are used to obtain homogenized properties for assemblies. Typically, the homogenized cross sections are calculated while preserving reaction rates [30].

$$< \Sigma_{x,g} >= \frac{\sum_i \phi_{g,i} \Sigma_{x,g,i} V_i}{\sum_i \phi_{g,i} V_i} \tag{3.20}$$

However, direct-weighted cross-sections do not allow to target heterogeneous reaction rates except in cases where the output geometry is homogeneous [4]. If an assembly is homogenized into a simplified heterogeneous geometry, like the one shown in Figure 3.3b representing a pin-by-pin homogenization, equation 3.20 is not valid. In fact, equation 3.20 does not preserve the fluxes and currents at the interfaces of the various nodes in the homogenized geometry. [31].

Equivalence techniques maintain the results obtained from heterogeneous transport calculations, which serve as reference computations.

Both Equivalence Theory (ET) and Generalized Equivalence Theory (GET) allow flux discontinuities at the interfaces within the homogenized geometry [31]. This approach ensures the preservation of reaction rates in each node and currents at the node interfaces. For each node i and interface $*$, an equivalence factor is introduced, defined as:

$$f_i^* = \frac{\phi_i^*}{\hat{\phi}_i^*} \tag{3.21}$$

Where $\hat{\phi}_i^*$ is the flux distribution at the interface $*$ from the homogenized geometry calculation, while ϕ_i^* is the one determined from the reference solution. The interface conditions are modified so the product between the flux and the equivalence factors must be contin-

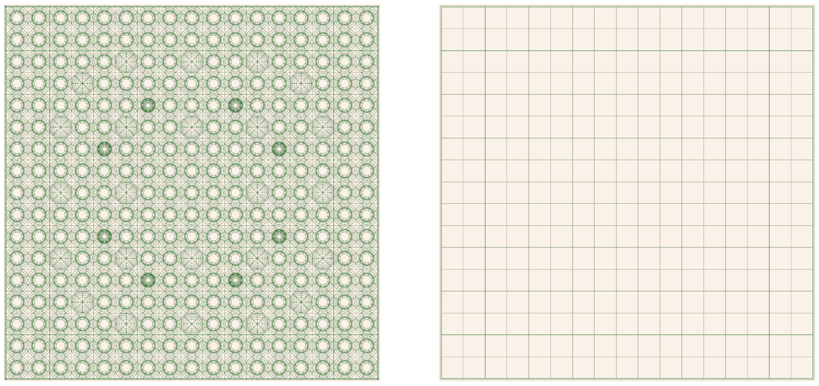

(a) Flux geometry (b) Pin-by-pin geometry

Figure 3.3 Pin-by-pin homogenization. Neutronic properties of regions in Figure (a) are homogenized into the geometry of Figure (b)

uous. Exact values of the equivalence factors can be found for any value of the diffusion coefficient for each node. The ET method iterates then on the diffusion coefficient values until the equivalence factors for each interface are the same. The resulting coefficients are called heterogeneity factors, and they allow to solve the homogenized diffusion equation while keeping the K_{eff}, surfacic currents, averaged fluxes, and reaction rates consistent with the heterogeneous reference solution. The GET method avoids this iterative strategy by taking advantage of the fact that exact equivalence factors can be determined for any diffusion coefficient value. Accordingly, no iteration is performed, fluxes are kept discontinuous, and conventional flux-volume weighted diffusion coefficients are used. In this situation, the equivalence factors are called discontinuity factors. Other techniques are also available, like the supergomogénéisation (SPH) equivalence technique [4]. Discontinuity factors obtained in 1D geometries can be transformed into SPFH factors without iterations [39].

Reflector models are approached differently than assembly models. While in assemblies, the key quantities to be preserved are reaction rates and leakages, for reflectors, the primary interest lies in their reflective properties or, more broadly, the reflector's impact on the flux in the core region. The behavior to be retained lies entirely outside the medium for which equivalent properties are sought.

Two reflector models commonly used in the industry are introduced in Annex A. Both rely on reference calculations based on a one-dimensional heterogeneous slab geometry approximating the average reflector profile. The flux is sustained by feeding assemblies whose properties are typically derived from the homogenization of a complete assembly.

3.3.1 Comments on reflector homogenization

One-dimensional models are especially effective for light reflectors, such as those found in Generation II PWR reactors. These reflectors, primarily composed of water, exert a strong moderating effect on neutrons. Hence, light reflectors return to the reactor a flux that has been significantly thermalized. These thermalized neutrons react with the fuel near the core's periphery, having minimal influence on the flux behavior in the core's inner layers.

Considering heavy reflectors, like those in VVER-1000 or EPR reactors, water constitutes only a small portion of the total volume. Given the reduced moderating effect, the neutron spectrum redirected towards the core is much faster. As a result, the neutrons scattered back into the core will penetrate deeper, thereby influencing the neutron flux even in the innermost regions. While inaccuracies in modeling light reflectors mainly affect the outermost pins of the core, for heavy reflectors, inaccurate modeling could introduce discrepancies across larger core areas.

Furthermore, 1D models do not capture the geometric complexities of heavy reflectors. Reflective properties vary across different points in the periphery. Some approaches sidestep this issue by executing multiple 1D calculations and adjusting the reflector's composition and geometry to better approximate specific regions. Recently, the CAMIVVER project also addressed two-dimensional reflector modeling using approximations performed in classical schemes (cylindrical and cartesian reflector models), showing important local discrepancies in pin-by-pin reaction rates at the interface between the core and the reflector compared with a reference calculation [8].

The present work makes some headway toward improving heavy reflector modeling by providing easy access to detailed reflector geometry descriptions.

3.4 Creation of the multi-parameter data library

Core calculations are fed with the information derived from lattice calculations, in particular condensed and homogenized neutronic properties of materials (cross sections, diffusion coefficients). The reactor's behavior must be studied at nominal, accidental, and perturbed conditions. For instance, one should consider variations of local and global reactor parameters like fuel and moderator temperature or variations of boron concentration. The reactor must be simulated for its operational lifetime; hence, isotopic evolution due to burnup must be considered.

Specific cross-section and diffusion data are required for each operational condition. Lattice codes provide a dataset, called Multi-Parameter Output (MPO) in APOLLO3®, from which it is possible to extract, through interpolation, the neutronic parameters for any combination of global and local reactor parameters.

Figure 3.4 provides a schematic representation of the calculation sequence, where only three reactor parameters are represented; however, typical applications usually have more. Isotopic evolution of materials is accounted for by performing burnup calculations, that is, computing materials' nuclear properties at nominal conditions as a function of the burnup. This is done by dividing the considered period into discrete time steps. At the end of each step, the isotopic composition of the materials is updated by solving the Bateman equations, assuming that the power remains constant. The lattice calculation is restarted with the new isotopic compositions, and new properties are computed and stored in the database. Hence, a burnup calculation is a sequence of elementary lattice calculations performed at nominal conditions, where the isotopic composition of materials derives from the information from a previous elementary calculation.

Several branch calculations are performed for each burnup step to account for variations in other reactor parameters [32]. The reactor parameters, such as fuel temperature, boron concentration, and moderator density, are perturbed from their nominal conditions using the current isotopic composition at specific burnups. Elementary lattice calculations are then performed for various combinations of the perturbed values of reactor parameters, and the resulting properties are added to the database.

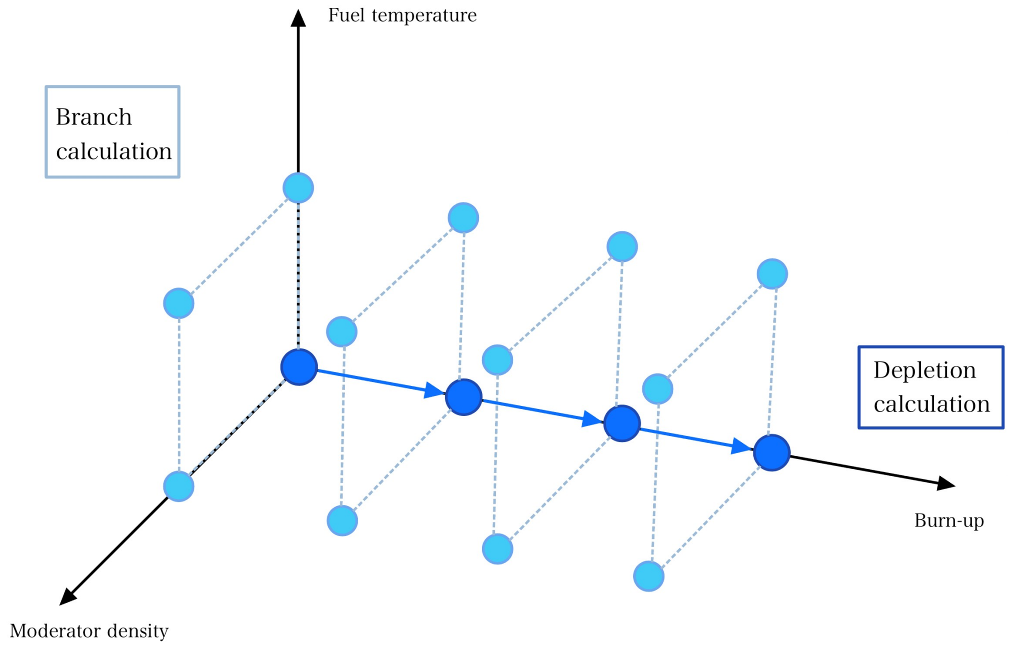


Figure 3.4 Schematic representation of the calculation tree for a generic MPO.

CHAPTER 4 NEMESIGeo_R library structure

The current chapter describes the structure and architecture of the Python library tool, NEMESIGeo_R, developed during the present work, that allows to easily model 2D radial reflector geometries. The library has been developed with the aim of being compatible with any reflector type, heavy or light, and any core configuration with cartesian or hexagonal lattice. Accordingly, the library must be able to reproduce any kind of real reflector geometry, from full configurations as EPR, PWR and VVER reflectors, to reflectors compatible with numerical minicore benchmarks, such as the KAIST-like heavy reflector [1] or the VVER minicores [33, 34] proposed in the CAMIVVER project.

Producing reflector geometries is made easier with the library as compared to ALAMOS. It provides the user with great flexibility when defining reflector modeling. For instance, users can choose refinement parameters and apply self-shielding properties to each medium.

The geometries generated by the library are exported to feed calculations. Depending on the solver used, the geometries to be provided will be different. For example, a lattice code will require a refined geometry. In contrast, a Monte Carlo code such as TRIPOLI4® does not require refinement but needs specific precautions to be compatible with certain geometries. If the APOLLO3® code is used, two approaches are available. The NEMESIGeo_R library can prepare a full core geometry suitable for APOLLO3® by merging the reflector with a core defined using the NEMESIGeo_C library. Alternatively, the reflector geometry can feed APOLLO3® calculation using a native core geometry. This latter option was used at the beginning of the present work, when NEMESIGeo_C was still unavailable.

The development of the library requires a good knowledge of the main engineering components and the general shape of reflectors. In the library, these components are grouped in the Item class and can be represented by a superposition of some relatively simpler shapes (i.e., a water channel with annuli is nothing but a superposition of circles) made out of one region only that are organized within the OneRegionItem class. In order to successfully assemble the geometry, some specific functions, each with a particular aim, are required. These methods are grouped into four classes and are used as "building tools" during the reflector building process. Other classes are also available for item positioning purposes and for producing an output homogenized geometry. Finally, a Reflector class merges all the information the user provides and builds the desired geometries.

Accordingly, an overview of the most common reflector shapes is given, focusing on western PWRs (with light and heavy reflectors) and VVER models. Then, an analysis of the user needs is provided to determine the library's desired capabilities. The library architecture is then presented, describing the most used classes and procedures. Finally, Appendix B.1 gives a practical library usage example.

4.1 Reflector components

Even if significantly different, both heavy or light reflectors share some common features that allow one to devise a way to automate the building process. A review of the most common reflector geometries is presented below to identify main reflector components and group them into classes with some common properties.

The current section aims to overview the reflectors used in most currently operational power plants. First, a brief overview of western PWR light reflector is given, then heavy reflector geometries are presented for both cartesian (EPR) and hexagonal (VVER) cores. The goal is to identify similar patterns in these geometries and to provide a list of engineering components that belong to reflectors.

4.1.1 Classical PWR light reflector

The shape of a PWR reflector is given in Figure 4.1. Pressure vessel internals are mainly composed of two water regions separated by an envelope. Thermal shields, used to limit neutrons from damaging the vessel, are distributed close to the envelope in the second water crown. The core is separated from the first water crown by a thin baffle.

Accordingly, the reflector can be represented by drawing four concentric circles, representing the vessel and the envelope's inner and outer perimeters, inserting a geometry replicating the shape of the reactor's core, and finally, appending the thermal shield.

4.1.2 Heavy reflector layouts

Heavy reflectors are different from light ones by their composition and complexity. These are mainly composed of steel, the water representing only a small fraction of their total volume. The baffle is way larger than the one found in light reflectors. Its radial section presents several cooling water channels and structural elements, such as tie rods and keys, increasing the complexity of these structures.

Reference geometries for cartesian (EPR) and hexagonal (VVER-1000) cores are given in Figures 4.2, 4.3 and 4.4. In both cases, the reflector structure is composed of circular regions, representing the vessel, downcomer, and barrel, and an inner region reproducing the shape

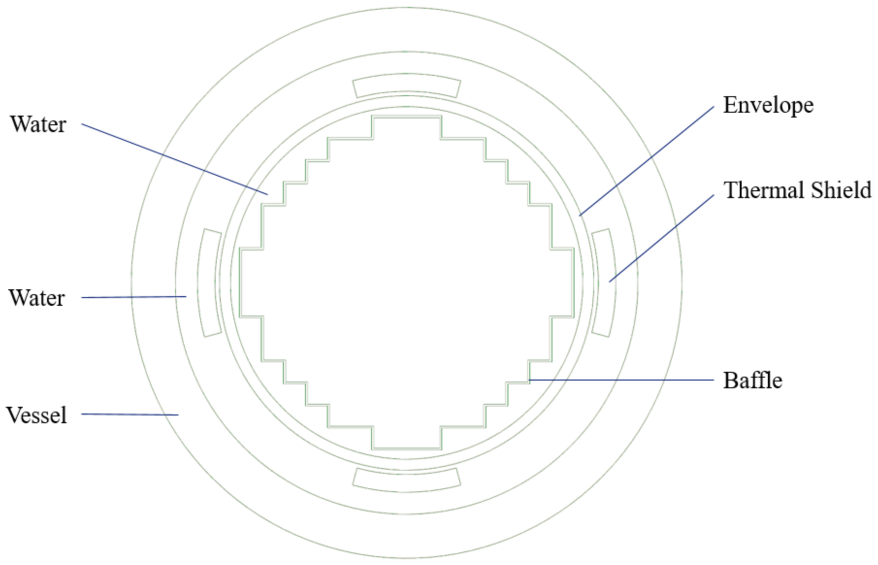


Figure 4.1 Layout of a PWR reflector

of the core. The VVER reflector's structure is more complex due to the presence of a basket groove region, modeled as a mixture of water and steel, whose geometry is far from being accessible by using native geometries and requires several parameters to be defined.

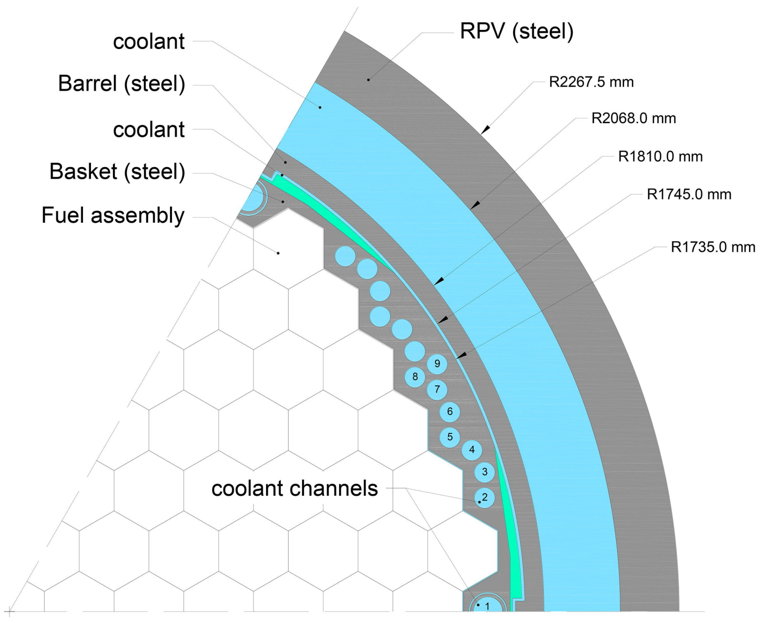
The two reflectors also differ from the present engineering components. Several small water channels cross the EPR baffle, while a few large channels pass through the VVER reflector. EPR models also account for structural elements supporting the baffle, such as tie rods and keys.

4.2 Analysis of the library requirements

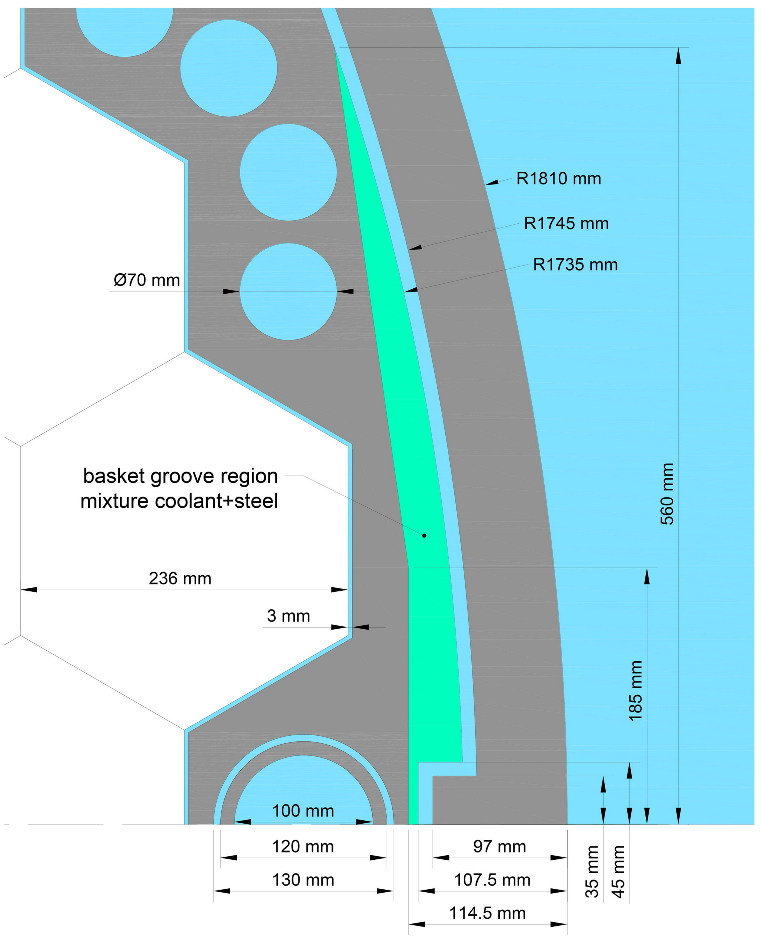
By observing the reflector's structures, it is possible to devise a way to automate the building procedure. In all observed cases, the reflector's structure consists of circular regions surrounding a core-shaped region. It is therefore possible to first outline the reflector shell by overlaying relatively simple geometries. The engineering components can then be superimposed onto the shell.

From the examined models, we can list the sub-geometries required to represent the reflectors of interest. These sub-geometries are:

- Circular regions for reflector's shell
- Core-shaped region (for both hexagonal or cartesian core)



(a) Sixth of VVER-1000 reflector [6]



(b) Core basket details [6]

Figure 4.2 Layout of a VVER-1000 reflector by symmetry [6]

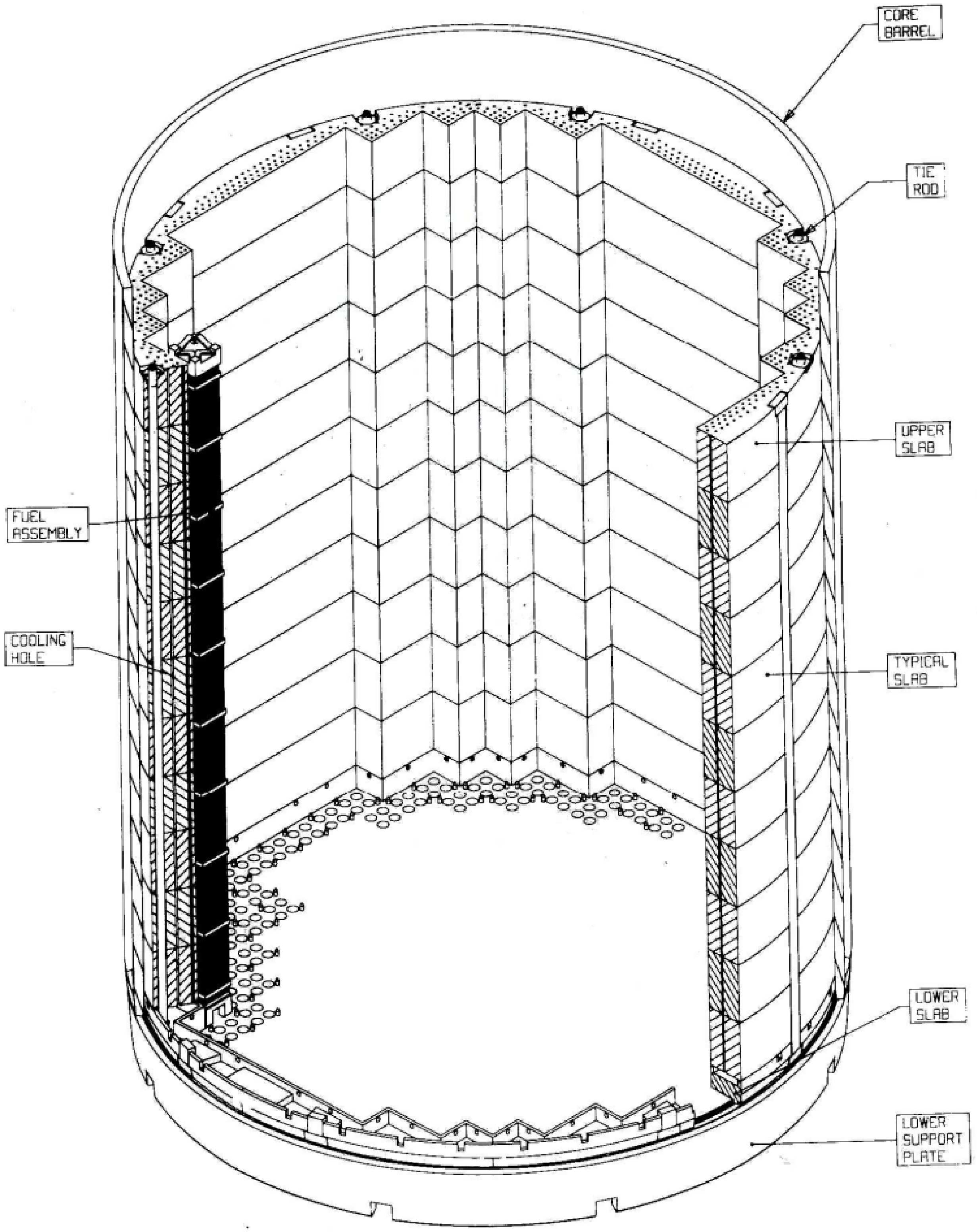


Figure 4.3 Layout of an EPR reflector [7].

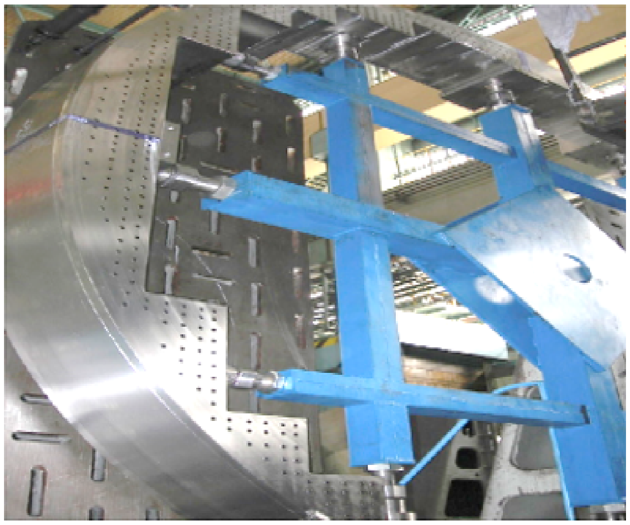


Figure 4.4 Picture of an EPR reflector's baffle

- Water channel
- Key
- Tie rod
- Groove region

To draw the reflector structure, the user must first define all the sub-geometries listed above. For this reason, each of these sub-geometries is described in a specific `Item` class and can be drawn starting from elementary geometries composed of a single region, such as circles and rectangles, represented in a `OneRegionItem` class.

The user needs to define materials and temperatures in each region that will be created. ALAMOS can do this by creating a *material* and *temperature* fields and setting the corresponding values in each region. Here, the value for these two fields is just a keyword the user defines that will be associated with a real mixture composition or temperature in the APOLLO3® input files. The user also needs to define another field associated with the self-shielding property. Each region having the same self-shielding property will be treated in the same way for self-shielding purposes.

Each region must be properly refined according to the user's needs. Refinement parameters can differ from region to region; hence, for each region, the user must be able to specify the corresponding refinement parameters. This feature is helpful to perform sensitivity studies related to geometry refinement.

Finally, the user needs to export all the geometries required to run a lattice calculation. Different geometries are required for flux and reflector self-shielding calculations. These geometries are connected by using the self-shielding property field. After flux calculation, the user may want to produce homogenized cross-sections. Accordingly, the library must also produce a homogenized reflector geometry having the same format used for assembly homogenization (pin-by-pin, quarter of assembly, assembly-by-assembly with or without water gap). A TRIPOLI4® geometry must be available if needed. Each geometry must be cut according to the symmetry of the problem.

4.3 Library architecture

The present section presents the main classes defined in the library and how they interact to produce the desired geometry. Initially, the `Item` and `OneRegionItem` classes, which describe geometric patterns of reflectors, are presented. Then, classes to store data related to item positioning and output options are introduced. Finally, the so-called "building tools", the `Reflector` class, and the most important related procedures are presented.

4.3.1 OneRegionItem

The basic geometries that will be used to assemble the engineering components of the reflector are grouped into different classes according to their shape and, hence, the parameters required to define them properly. The shapes considered here are all made out of one region and are:

- Circles
- Polygons
- Rectangles
- Cartesian and hexagonal core shapes

The classes corresponding to these basic shapes are grouped under the name of one region item class. Figure 4.5 gives a UML diagram for these classes. Generally, the user will not directly use these classes; he will instead use some higher-level classes already tailored to draw the engineering and structural components of the reflector and grouped under the name of item classes.

The `OneRegionItem` classes have an implemented method capable of drawing their geometry. Additionally, each instance should be named, allowing for easy identification of the

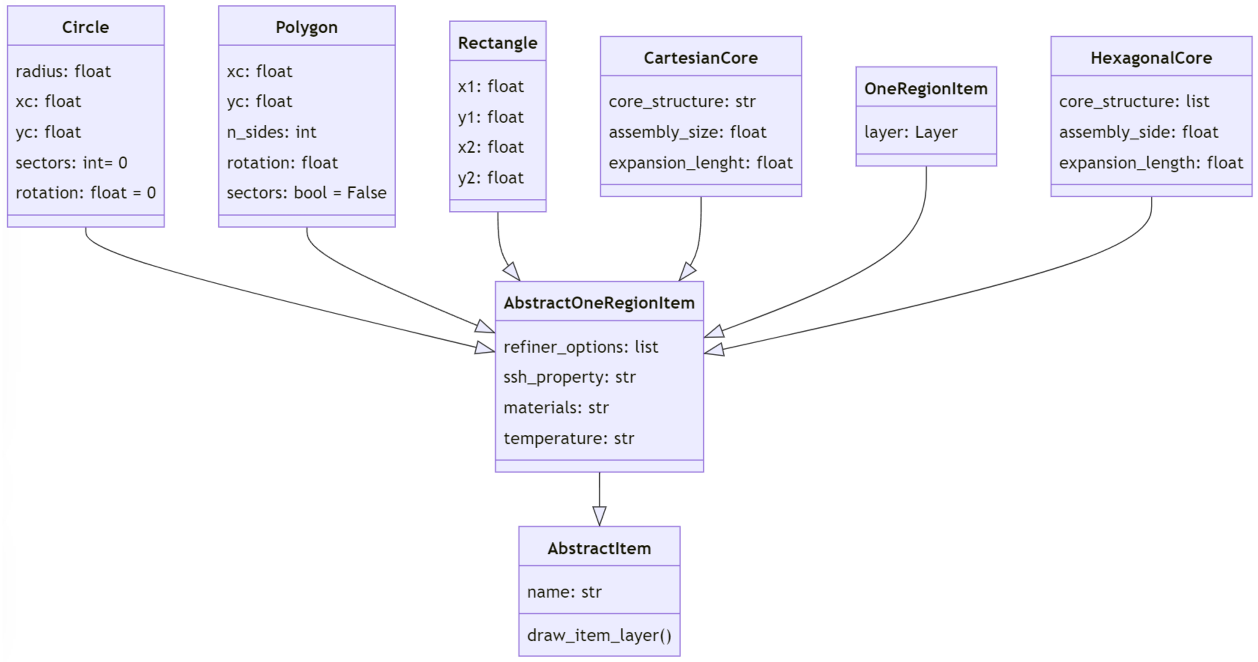


Figure 4.5 UML class diagram for the OneRegionItem class

corresponding layer on ALAMOS. This feature, common also with the `item` classes, is implemented by letting these classes inherit from an `AbstractItem` class, dictating that each class will have a `name` attribute and a method for the representation of the geometry called `draw_item_layer()` .

Since `OneRegionItem` represents the most fundamental geometries handled by the library, the information related to the material composition, temperature, self-shielding properties and refinement options is assigned directly when an `OneRegionItem` object is instantiated. In this way, the building process will consist of overlapping simple shapes, resulting in a final geometry in which each region is already defined with its fields. As a result, the classes will inherit from an abstract class named `AbstractOneRegionItem` that adds new attributes used to characterize the layer fields.

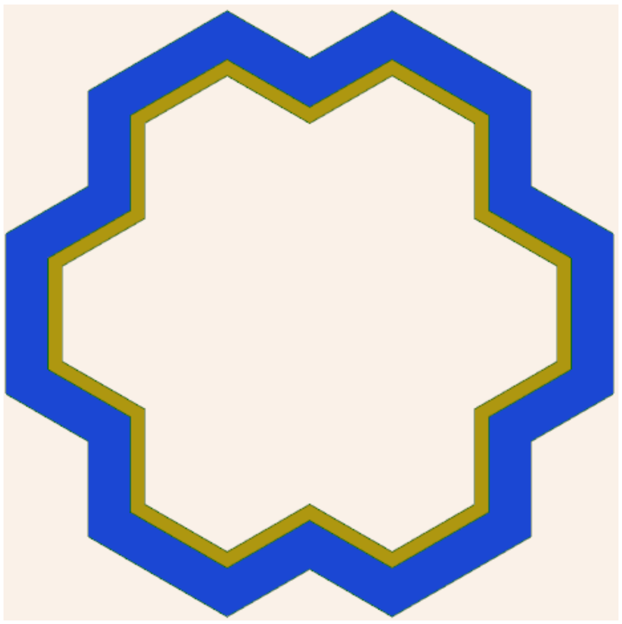
Circle, Polygon and Rectangle classes

ALAMOS already has functions that allow representing circles, polygons and rectangles. Accordingly, the methods used to draw the corresponding layers are by far the simplest of the library, and the procedure is similar to what is shown in the code snippet presented in Section 1.1.1.

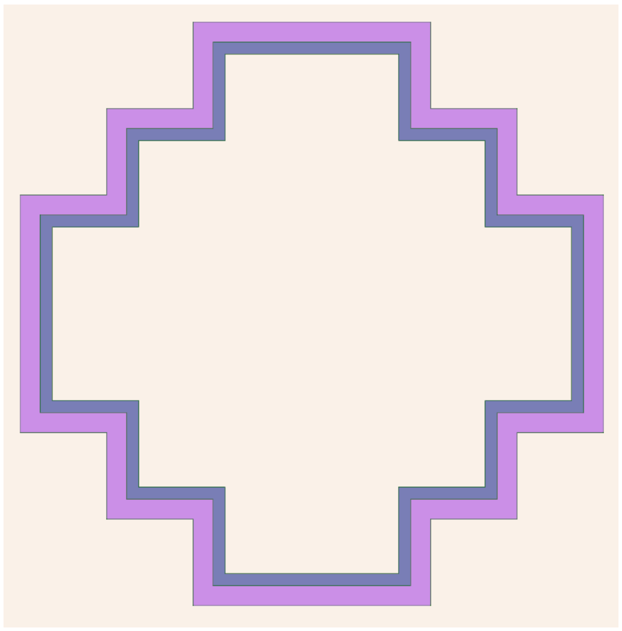
Core-shaped region classes

The core's shape defines the internal perimeter of the reflector. Depending on the core type, the shape can be based on cartesian or hexagonal geometries and is represented by the `HexagonalCore` and `CartesianCore` classes, respectively.

A mention should be given to the `expansion_lenght` attribute. This attribute unlocks the possibility for the user to produce some regions surrounding the core that replicate its shape while keeping almost everywhere a constant distance from the core, equal to the `expansion_lenght` attribute. This feature allows the user to produce a finer refinement in the regions closer to the core, as shown in the example provided in Appendix B.1, and also to assign some different self-shielding properties according to the distance from the core, as shown in Figure 4.6.



(a) Hexagonal core shaped regions



(b) Cartesian core shaped regions

Figure 4.6 Core shaped regions. Different colors represent different values for the self-shielding property field.

In order to produce these regions, the user only needs to provide a map of the core structure expressed as a string of characters for cartesian cores or a Python list object for hexagonal cores.

Generic class

If a shape different from the ones listed above is required, the user can make use of the `GenericOneRegionItem` class. The user only needs to produce the layer corresponding to the desired geometry and use it to instantiate an object of this class. In this way, the desired shape can be used to build the reflector since the defined object will be treated the same way as the other predefined geometries listed above.

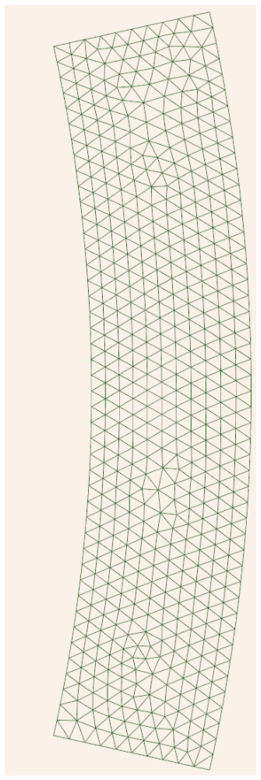
4.3.2 Items

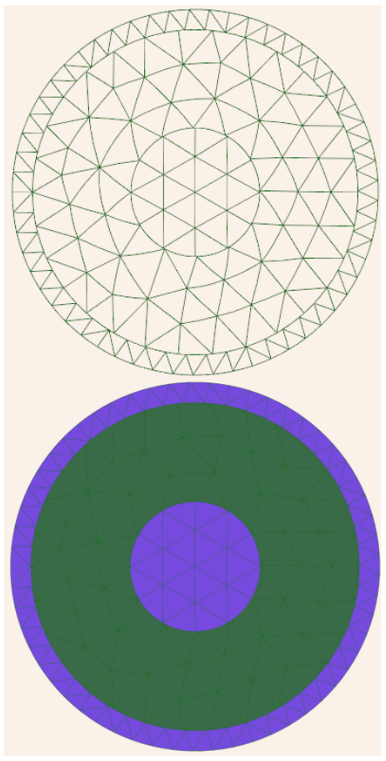
The `Item` class aggregates all the engineering components and structural elements listed in section 4.2. The initial step for users involves defining all the `Item` objects that make up the reflector geometry. A UML diagram of the `Item` class is presented in Figure 4.8. As noted, `Item` objects are essentially aggregations of `OneRegionItem` objects, inheriting shared properties from the `AbstractItem` class.

Structural components are represented by the `HexagonalCoreShapedRegions` and `Cartesian CoreShapedRegions` classes for the reflector's inner perimeter and `CircularRegions` class to represent the external layers. Core-shaped regions, depicted in Figure 4.6, aggregate either `HexagonalCore` or `CartesianCore` objects. The `CircularRegions` class is versatile in the sense that it can be used to model reflector circular regions but also any other unforeseen circular engineering components.

In VVER reflectors, the baffle is surrounded by the basket groove region, separated by a thin water gap. Due to the complexity of this region, several parameters are required to model this geometry. These parameters reside in the `WaterGapGrooveRegionOptions` and `BasketGrooveRegionOptions` classes, prerequisites before instantiating a `GrooveRegion` object.

The remaining classes detail the engineering components. Both `TieRod` and `WaterChannel` are specializations of the `CircularRegion` class, implying their structure is predefined, and users need only to specify certain parameters. The `ThermalShield` resembles an arc segment with a specified amplitude and width, while the `Key` takes the shape of two adjoining rectangles representing the key and the corresponding water gap. The key tangentially aligns with one of the reflector's circular regions. To preclude a minor gap between the key and these regions, the steel rectangle's external side mirrors an arc with a radius matching the corresponding circular region. The geometry described by each of these classes is given in Figure 4.7 .


(a) Thermal shield


(b) Tie rod

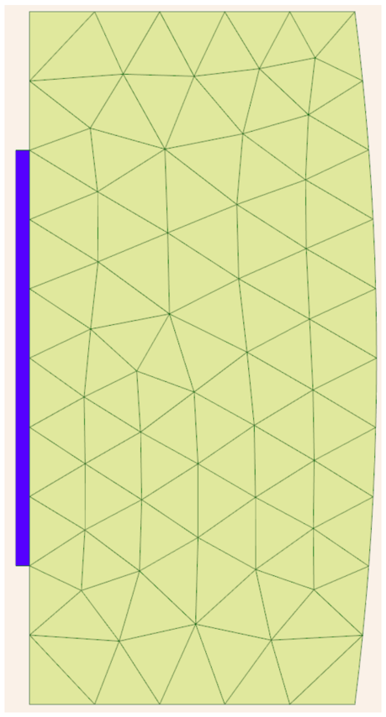

(c) Key

Figure 4.7 Common reflector engineering components. Water channels are not shown here because of their similarity with tie rods.

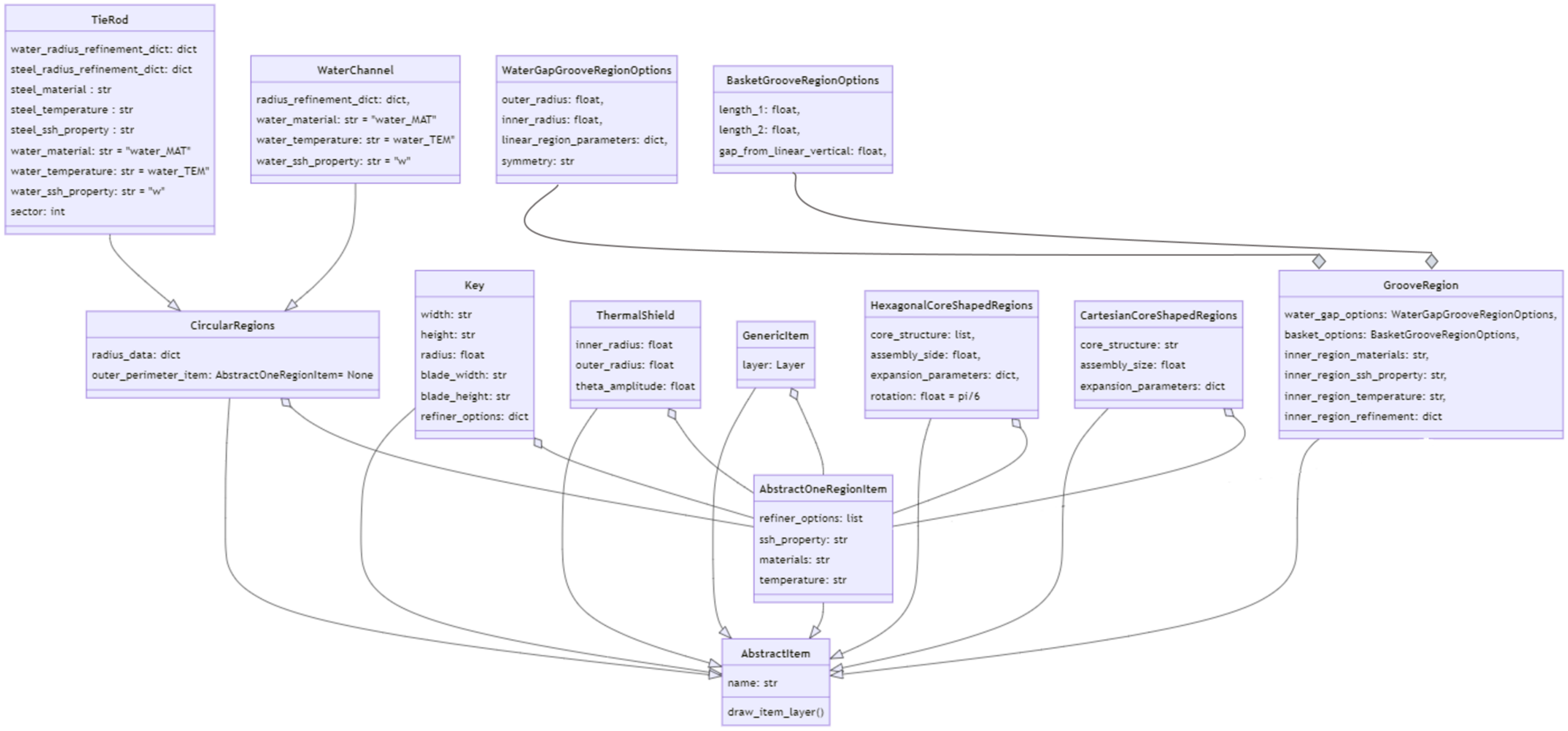
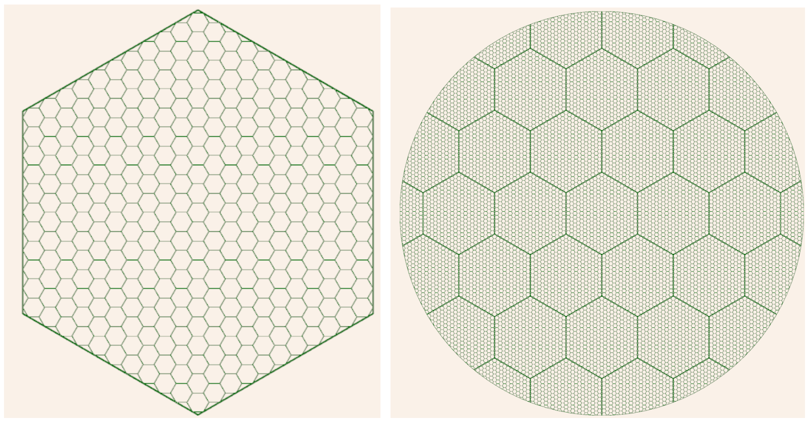

TieRod
water_radius_refinement_dict: dict
steel_radius_refinement_dict: dict
steel_material : str
steel_temperature : str
steel_ssh_property : str
water_material: str = "water_MAT"
water_temperature: str = water_TEM"
water_ssh_property: str = "w"
sector: int
WaterChannel
radius_refinement_dict: dict,
water_material: str = "water_MAT"
water_temperature: str = water_TEM"
water_ssh_property: str = "w"
WaterGapGrooveRegionOptions
outer_radius: float,
inner_radius: float,
linear_region_parameters: dict,
symmetry: str
BasketGrooveRegionOptions
length_1: float,
length_2: float,
gap_from_linear_vertical: float,
CircularRegions
radius_data: dict
outer_perimeter_item: AbstractOneRegionItem= None
Key
width: str
height: str
radius: float
blade_width: str
blade_height: str
refiner_options: dict
ThermalShield
inner_radius: float
outer_radius: float
theta_amplitude: float
GenericItem
layer: Layer
HexagonalCoreShapedRegions
core_structure: list,
assembly_side: float,
expansion_parameters: dict,
rotation: float = pi/6
CartesianCoreShapedRegions
core_structure: str
assembly_size: float
expansion_parameters: dict
GrooveRegion
water_gap_options: WaterGapGrooveRegionOptions,
basket_options: BasketGrooveRegionOptions,
inner_region_materials: str,
inner_region_ssh_property: str,
inner_region_temperature: str,
inner_region_refinement: dict
AbstractOneRegionItem
refiner_options: list
ssh_property: str
materials: str
temperature: str
AbstractItem
name: str
draw_item_layer()

Figure 4.8 UML class diagram for the Item class

4.3.3 Other classes

To complete the general framework, users need to specify the position and orientation of each engineering component. The positioning details for every component are stored in an `ItemPositioningInstruction` object. These objects possess an `instruction` attribute that describes the sequential rotations and translations to apply to each component. Users can instantiate an `ItemPositioningInstruction` object one at a time. However, if the same `Item` must be placed in multiple positions, such as for the `WaterChannels`, factory methods are available to produce lists of `ItemPositioningInstruction` directly.

Users can also define the format for the output geometry in an `OutputOptions` object. This feature allows for the automatic generation of output in pin-by-pin, quarter of assemblies, or assembly-by-assembly formats for cartesian geometries. This also considers the water gaps once the pins, assemblies, and water gap dimensions are specified. For hexagonal geometries, this feature still needs to be implemented. However, generating output geometries in hexagonal or non-standard Cartesian formats remains possible by providing the layer containing the output geometry for an assembly, like the one that can be generated using the lattice libraries. This latter situation is shown in Figure 4.9.



(a) Assembly homogenized geometry

(b) Output geometry for the full core with reflector geometry

Figure 4.9 Assembly homogenized geometry (a), produced by the `NEMESIGeo_C` library, to be given as input to `NEMESIGeo_R` generate the final output geometry (b)

4.3.4 Building tools

The previous section describes the toolkit provided to the user to define the reflector's structural and engineering components. This section introduces the most important internal procedures used during the building process, organized into four classes shown in Figure 4.10 to separate their functionality. As mentioned earlier, users must specify each element that constitutes the reflector by instantiating the corresponding objects. The reflector is constructed through sequential overlapping of geometries: first, the reflector structure is drawn, and then the engineering components are overlaid onto the reflector's shell.

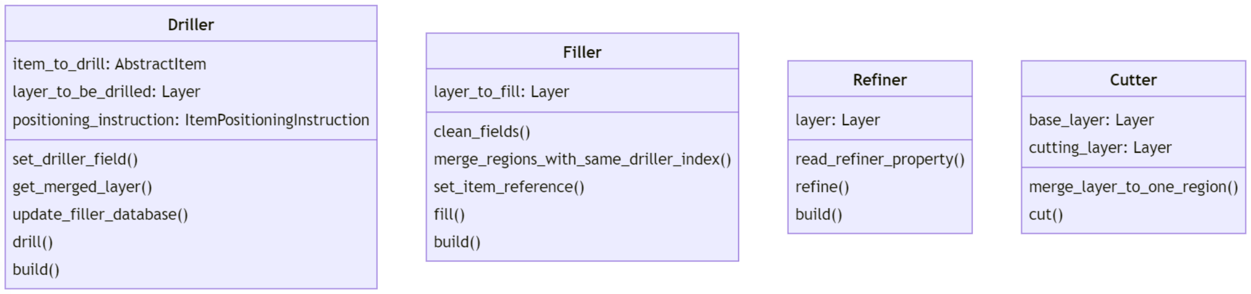


Figure 4.10 Building tools classes

The overlapping process of an item involves two steps: first, the item's perimeter is aligned with the main reflector's layer, and a reference is established between the newly created region and the item layer. Then, the reference is resolved, and the referenced layer is inserted into its hosting cell. The first procedure and all its subroutines are stored in the `Driller` class, while the dereferencing process is managed by the `Filler` class.

Each cell that is built contains information related to its refinement parameters. This information is stored in a specific string field called *refiner*. The `Refiner` class houses functions designed to read the refinement information for each cell and perform region refinement based on user specifications.

Finally, users may need to shape the reflector's geometry according to symmetry or geometric restrictions imposed by input calculation files. To address this requirement, a tool is provided and stored within the `Cutter` class. This class is also used to extract portions of the geometry to be used for self-shielding calculations.

For every class except the `Cutter`, a factory method called `build` is provided to avoid instantiating an object before executing every building function.

Driller

The class contains all the methods required to properly position the item in the reflector layer, as shown in Figure 4.11. The data required to instantiate a `Driller` object are the main reflector layer, representing the layer over which the item geometry will be drawn, the

`Item` object to be drawn, and data related to its position and orientation on the main layer. This latter information is stored in an `ItemPositioningInstruction` object, consisting of an ordered sequence of rotations and translations needed to accurately position the item within the reflector.

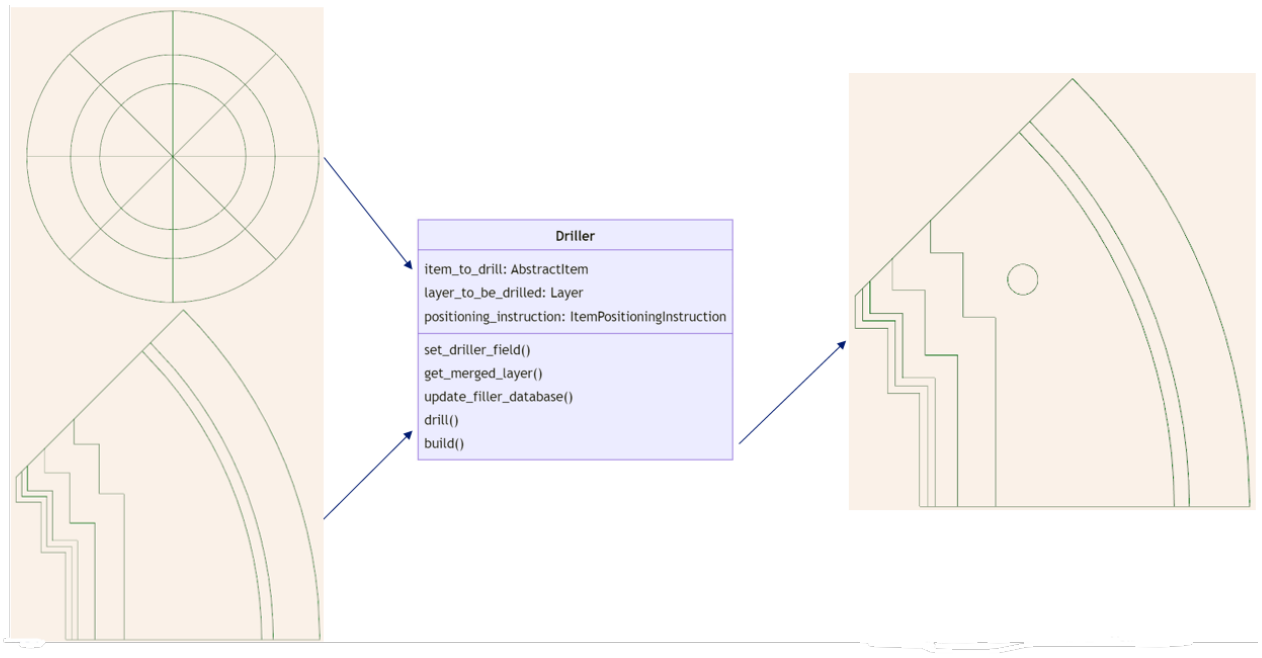

Driller
item_to_drill: AbstractItem
layer_to_be_drilled: Layer
positioning_instruction: ItemPositioningInstruction
set_driller_field()
get_merged_layer()
update_filler_database()
drill()
build()

Figure 4.11 Graphical representation of the drill procedure

Driller.drill() procedure A schematic representation of the `.drill` method is given in Figure 4.12.

Only the outer perimeter of the item object is drawn since dereferencing is performed later in the building procedure. For this reason, the first step is to produce a one region layer from the input `Item` by merging the item's geometry cells. If the `Item` to be drilled is a `OneRegionItem`, then the geometry is already considered as merged, and nothing is done at this stage.

Since the drill procedure is executed several times and with different items before filling the regions, one must keep track of the regions that need to be filled and their corresponding `item`. In the reflector layer, a field called `driller` is created, and each drilled region is associated with a unique value of that field, which allows linking the drilled region to the corresponding `Item`. The correspondence between the field value and the `Item` is stored in a dictionary class object attribute called `database`, which is updated every time that a `.drill` method is called and will be used by the `Filler` to perform dereferencing.

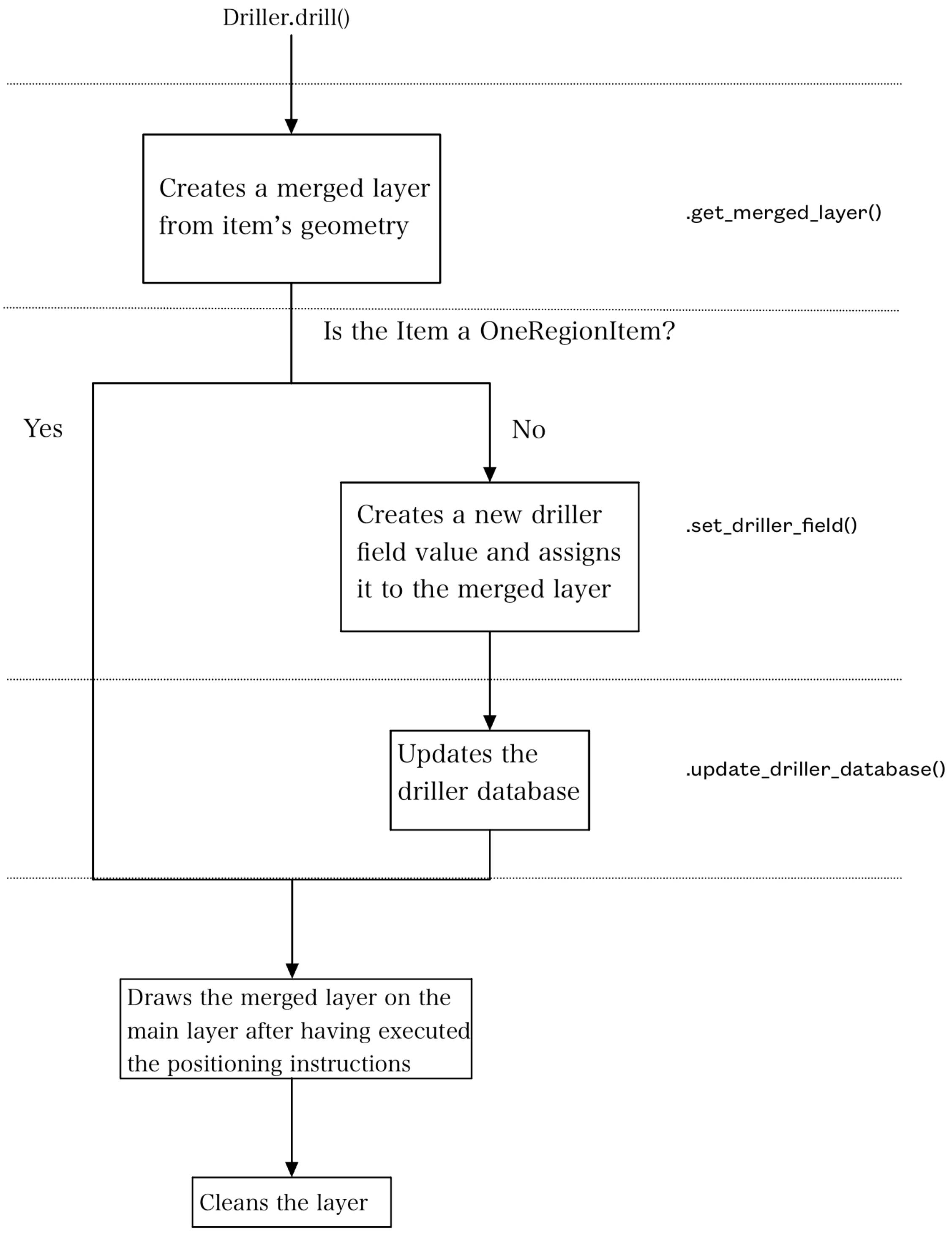

Driller.drill()
Creates a merged layer from item's geometry
.get_merged_layer()
Is the Item a OneRegionItem?
Yes
No
Creates a new driller field value and assigns it to the merged layer
.set_driller_field()
Updates the driller database
.update_driller_database()
Draws the merged layer on the main layer after having executed the positioning instructions
Cleans the layer

Figure 4.12 Schematic representation of the `.drill()` procedure

Finally, the merged layer is positioned on the main layer, and ALAMOS construction lines are cleaned.

Filler

The only data required for the `.fill` procedure to work are a previously drilled layer and the corresponding `database` stored in the `Driller` class. The `.fill` procedure operates as a secondary step to the `.drill` one, and deals with performing the dereference operation on all the regions that have been created after executing one or more `.drill` processes, as shown in Figure 4.13. Differently from the `Driller` class, where one object must be instantiated for each item at each position, a singular `Filler` object is responsible for appending all drilled item's geometries in their corresponding cells. The `.drill` and `.fill` sequences do not necessarily have to be called one after the other: once a `.drill` procedure has been carried out, it is possible to perform other manipulations on the layer, such as geometry refinement, before calling the `.fill` function.

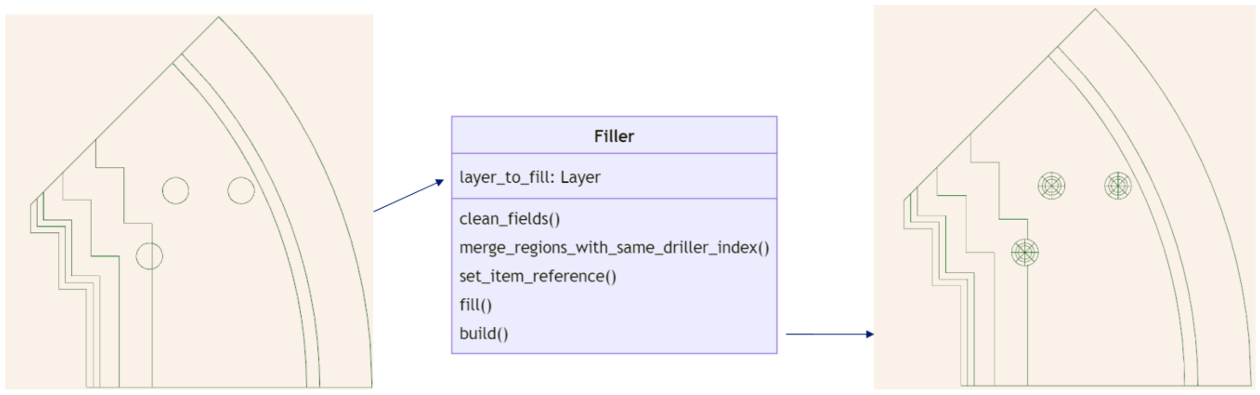


Figure 4.13 Graphical representation of the `.fill` procedure. The bottom left water channel is inserted so that it gets split in two by an expanded core region. The `.fill` procedure identifies these regions and merges them again before dereferencing the water channel.

Filler.fill() procedure In rare situations, some regions that have been drilled may intersect the edge of other regions already present on the main layer (see Figure 4.13). In this case, the regions in question will be divided into two or more cells, making it impossible to perform the dereference since dereference on geometries composed of multiple cells cannot be done in ALAMOS.

To overcome this problem and ensure that the `.fill` process works well, a pre-treatment of the main layer is carried out to reunite the cells with the same `driller` field index. Merging two adjacent cells is an ALAMOS functionality, but requires that the cells to be merged have the same values for each field. This requirement is typically not satisfied when a drilled region is split into different cells. Therefore, for each key in the driller `database`, representing all the non null `driller` field values, the following steps are undertaken:

- Cells with the current value of the *driller* field are identified.
- Default values are assigned to all fields except the *driller* field for each of these cells. This way, the cells will eventually have the same field values and can thus be merged.
- The identified cells are merged.
- The index of the single resulting cell, which has the current value of the *driller* field, is identified, and the reference with the layer of the corresponding `Item` is assigned.

At the end of this loop, all references will have been assigned to the main layer. It is therefore possible to resolve the reference relationship by inserting the geometries of the items in their corresponding cells.

The procedure concludes by deleting the information stored in the driller `database`. In this way, future calls to the `.fill` function will not try to insert already treated geometries.

Refiner

Reflector construction meshes, as they are, turn out to be too large to perform flux calculations; therefore, it is necessary to refine them. The ALAMOS library has proactively addressed this by introducing a feature that facilitates the triangulation of specific regions, recognizable through their cell indices. However, as it is aimed to automate the geometry generation process further, maintaining a track of the dynamic cell indices — which could alter with each layer operation — poses a significant challenge.

To counteract this, a *refiner* field has been established, embedded with all the essential parameters required for the cell refinement. This field derives its value indirectly through the user-defined `refiner_options` attribute of `Item` classes, streamlining the automation process by obviating the need for individual cell index tracking. Consequently, it is possible to refer to each cell's respective *refiner* field value to execute the refinement.

.refine() procedure The process, depicted in Figure 4.16, proceeds by iterating through each cell in the layer, looking for cells with a non-null *refiner* field value. When a region of that type is found, its field value is deleted after being saved.

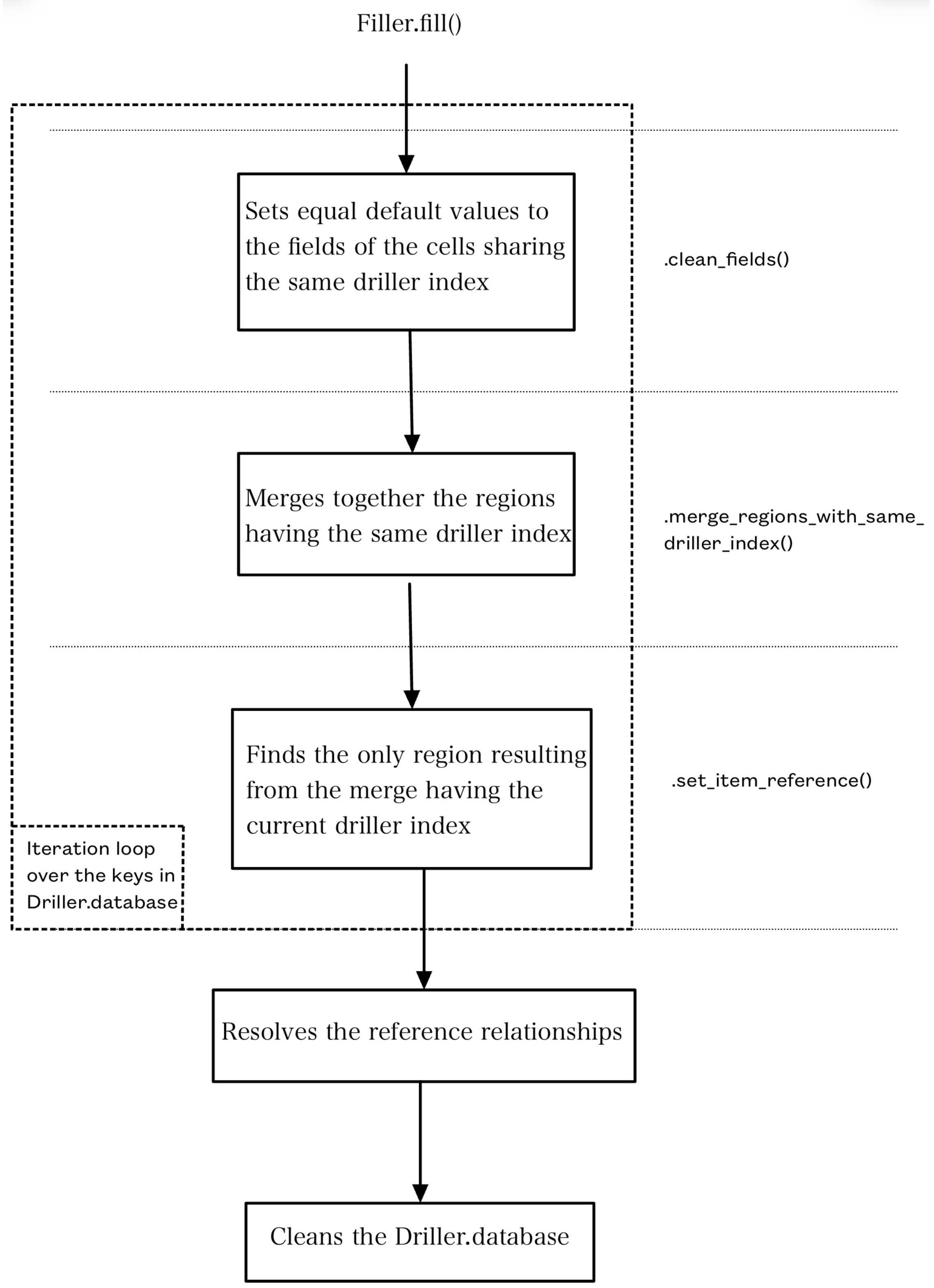

Filler.fill()
Sets equal default values to the fields of the cells sharing the same driller index
.clean_fields()
Merges together the regions having the same driller index
.merge_regions_with_same_driller_index()
Finds the only region resulting from the merge having the current driller index
.set_item_reference()
Iteration loop over the keys in Driller.database
Resolves the reference relationships
Cleans the Driller.database

Figure 4.14 Schematic representation of the `.fill()` procedure

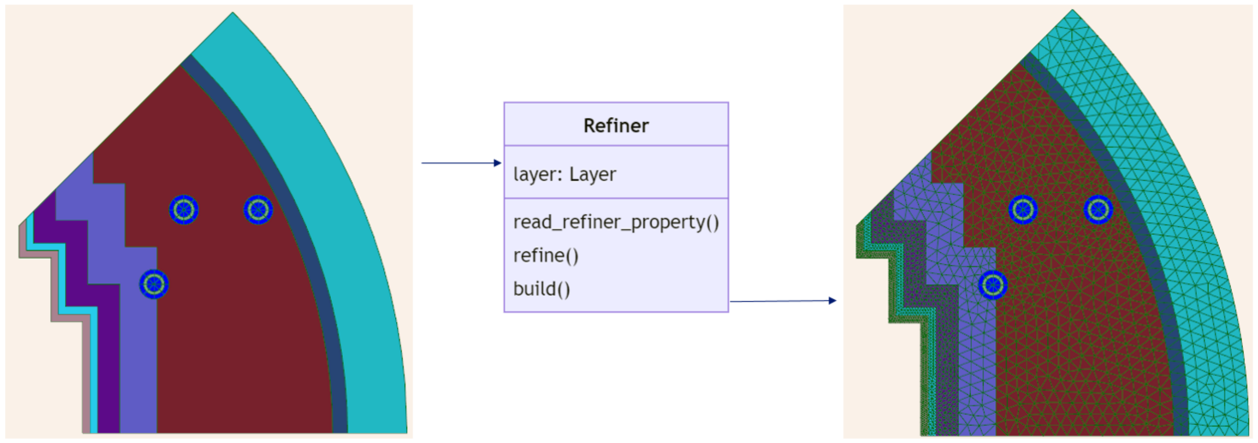


Figure 4.15 Graphical representation of the refinement procedure. Each colored region can be associated with different refinement parameters.

The values of the *refiner* field are text strings interpreted using the `read_refiner_property` function, storing the parameters used for refinement. The cell is refined, and the iteration continues over the cell indices. To refine the cell, a function already implemented in ALAMOS is used. This only requires three parameters, which are the same that the user must provide when defining the items.

With certain parameters, the automatic mesher may encounter an error. In this case, the user will receive information on the terminal to identify the cell and an invitation to change the refinement parameters or to reboot the software (often, this latter procedure solves the problem).

Since refining a region creates many new ones, a cell may change its index after the refinement of another cell has been completed. For this reason, following the iteration over the cell indices, some regions may have not yet been refined. Therefore, a check is performed to ensure that no remaining regions are to be refined. If no region is detected, then the refinement process concludes. Otherwise, the loop over the indices of each layer cell is carried out again.

Cutter

The `Cutter` class contains a useful method to cut a base layer's geometry according to another layer's shape, given by the `cutting_layer` attribute. This function can be used to trim portions of the reflector for self-shielding calculation purposes, or to tailor the reflector based on its symmetry as shown in Figure 4.17. The procedure is based on a pre-built ALAMOS function that allows to intersect two layers with each other. Since no elements need to be added to the geometry here, `cutting_layer` must be made out of only one region. The first

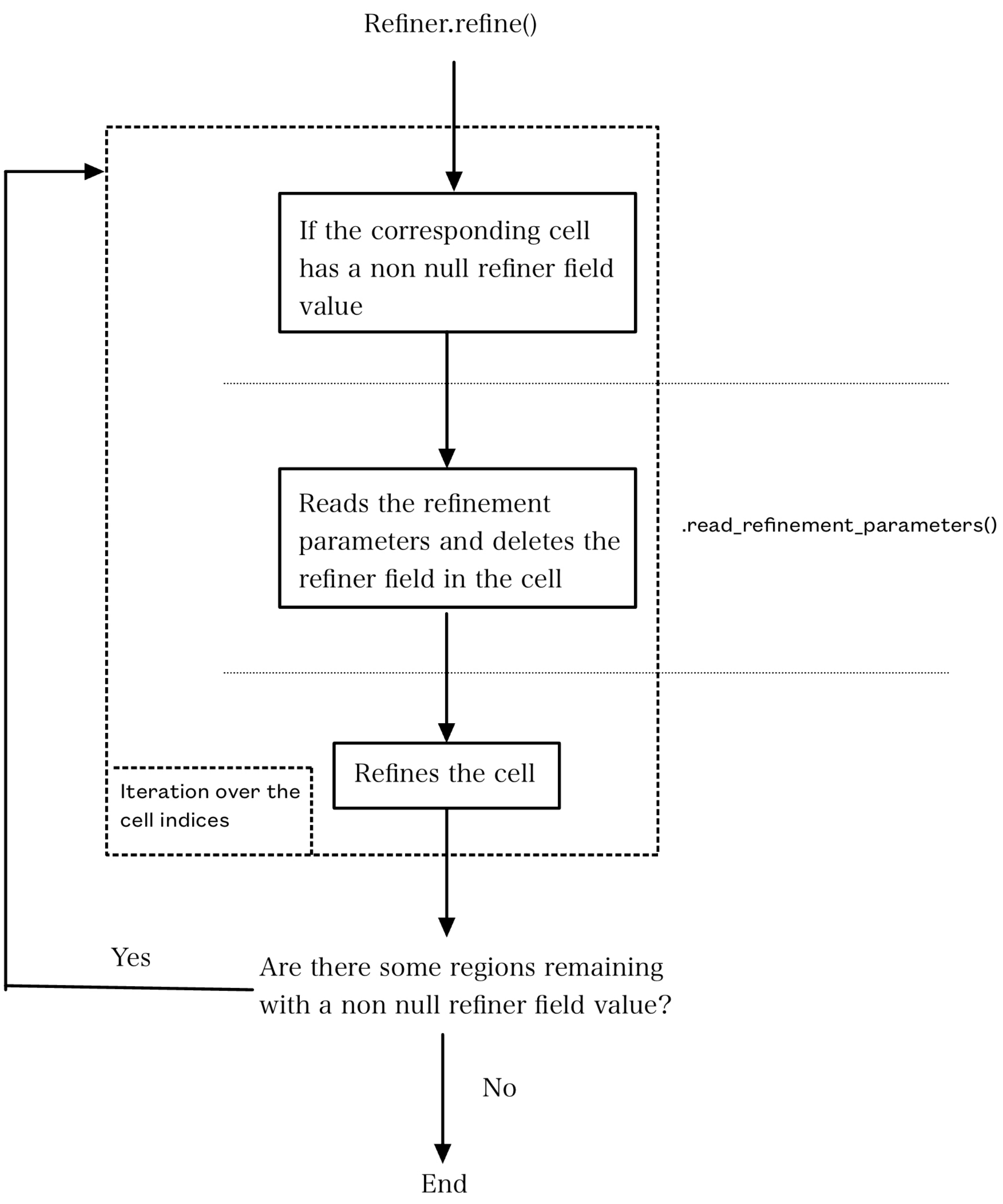

Refiner.refine()
If the corresponding cell has a non null refiner field value
Reads the refinement parameters and deletes the refiner field in the cell
.read_refinement_parameters()
Refines the cell
Iteration over the cell indices
Yes
Are there some regions remaining with a non null refiner field value?
No
End

Figure 4.16 Schematic representation of the `.refine()` procedure

step is to merge the `cutting_layer` similarly to what is done in the `.drill` procedure. Then, the intersection between the two layers is performed, and the resulting layer is returned.

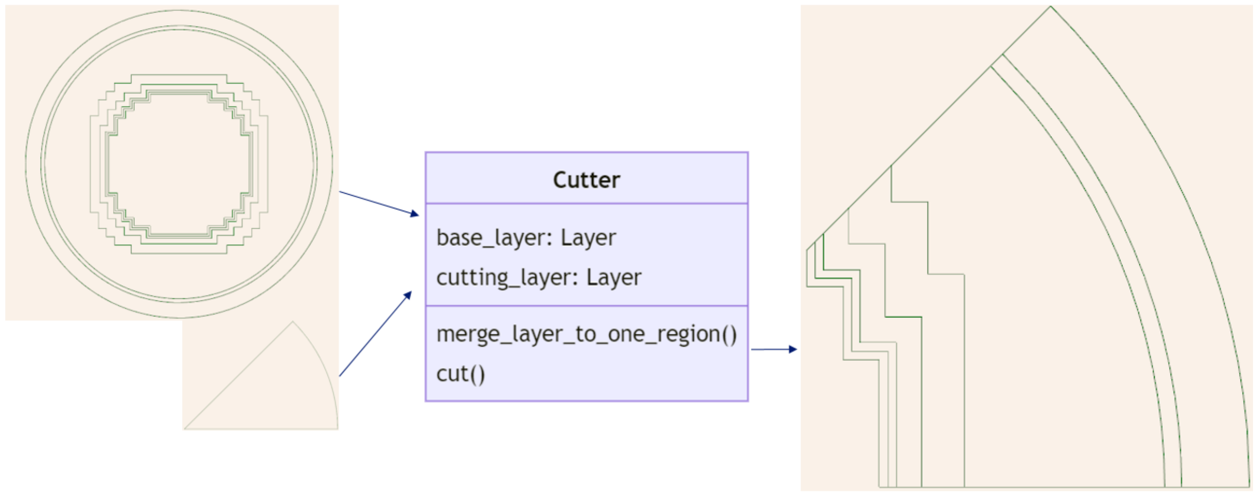


Figure 4.17 Graphical representation of the `.cut()` procedure

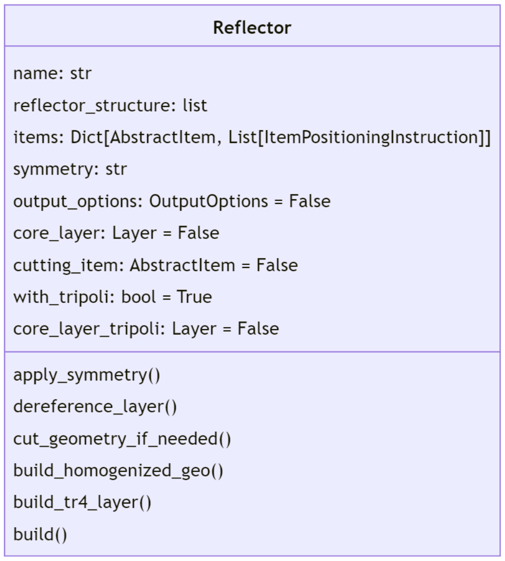
4.3.5 Reflector Class

The `Reflector` class manipulates all the objects and parameters the user defines to produce the reflector geometry. It stands as the final element to be defined, assembling all previously defined elements to reproduce the reflector's geometry. Figure 4.18 provides a simplified UML diagram.

The shell of the reflector comprises all objects that define its geometry, excluding the engineering components. These structural elements are the foremost to be placed into the principal layer, proceeding through a series of `drill` and `fill` operations. Therefore, the `reflector_structure` attribute should be a systematically ordered list, where the initial elements in the list are to be the first ones to be placed. Prioritizing larger objects at the beginning is imperative, as ALAMOS permits geometry construction solely through a top-down approach, making it unfeasible to draw around pre-existing objects.

Engineering components are introduced in the `items` variable. This is constituted as a dictionary where for every `Item`, it yields the corresponding list of `ItemPositioningInstruction` objects. Once all the items have been drilled, the reflector layer is refined. The procedure ends after applying the `.fill()` procedure over the engineering components. A simplified sequence diagram for the `Reflector.build()` procedure is shown in Figure 4.19.

At first glance, the two aforementioned attributes may appear sufficient to portray the re-


Reflector

name: str
reflector_structure: list
items: Dict[AbstractItem, List[ItemPositioningInstruction]]
symmetry: str
output_options: OutputOptions = False
core_layer: Layer = False
cutting_item: AbstractItem = False
with_tripoli: bool = True
core_layer_tripoli: Layer = False

apply_symmetry()
dereference_layer()
cut_geometry_if_needed()
build_homogenized_geo()
build_tr4_layer()
build()

Figure 4.18 UML diagram of the reflector class.

flector accurately. However, users can tailor other parameters according to their needs. One of the options available is to manipulate the geometry by leveraging the reflector's symmetry to cut it accordingly, hence reducing the number of regions. Additionally, they can choose to cut the geometry based on the shape of another layer. Beyond shaping the geometry, users can create various geometries to be used for different purposes, such as output homogenized geometries or TRIPOLI geometries. These choices empower users to work with various configurations to suit their project requirements.

In instances where the user prefers to generate a geometry containing both the core and the reflector directly, it is feasible to supply the respective core layer for both the main geometry and a potential geometry for TRIPOLI. The resulting layer can then be easily exported and used as an input file for calculations.

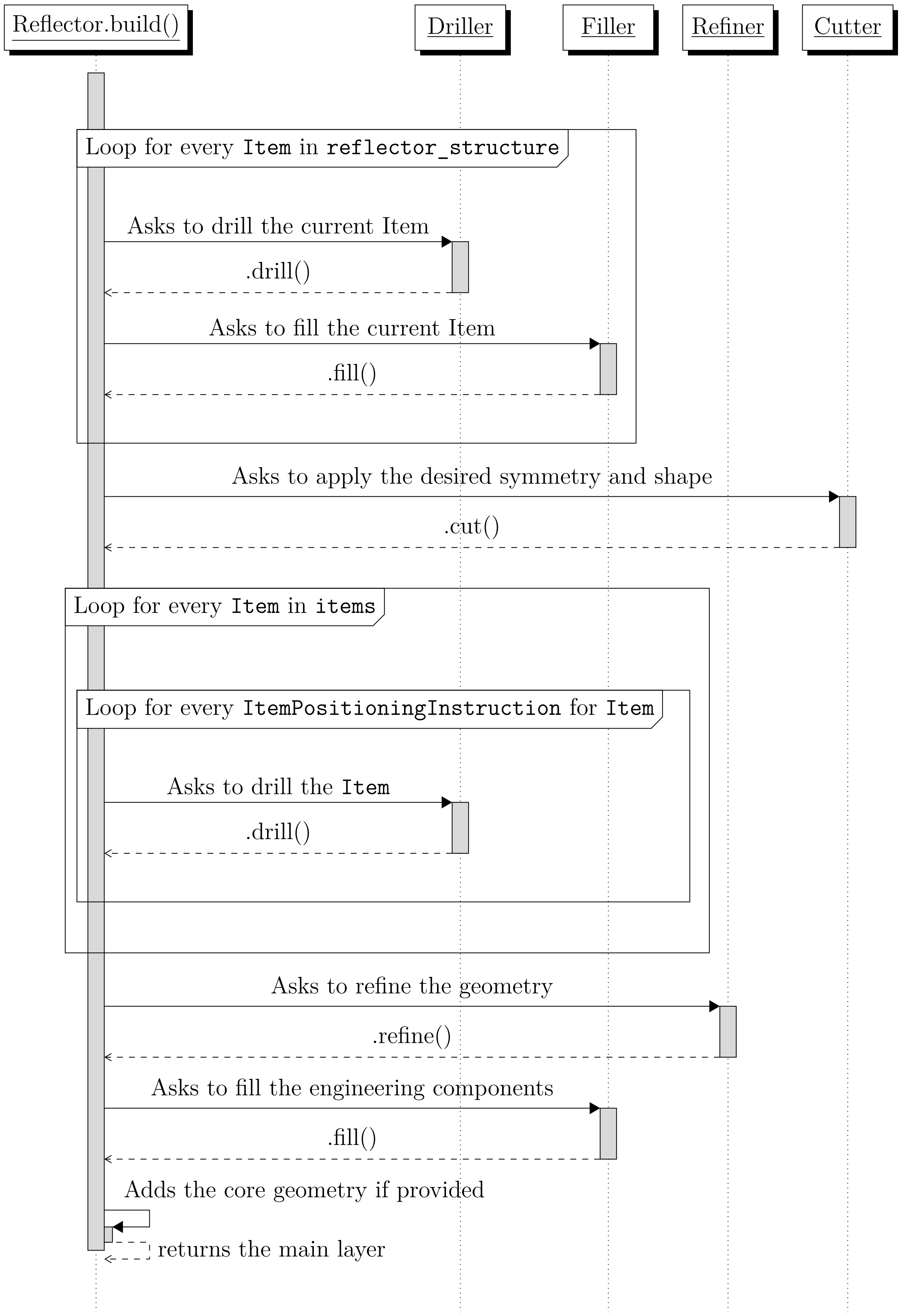

Reflector.build()
Driller
Filler
Refiner
Cutter
Loop for every Item in reflector_structure
Asks to drill the current Item
.drill()
Asks to fill the current Item
.fill()
Asks to apply the desired symmetry and shape
.cut()
Loop for every Item in items
Loop for every ItemPositioningInstruction for Item
Asks to drill the Item
.drill()
Asks to refine the geometry
.refine()
Asks to fill the engineering components
.fill()
Adds the core geometry if provided
returns the main layer

Figure 4.19 Sequence diagram of the Reflector.build() procedure. The diagram does not include the procedures to build TRIPOLI or homogenized geometries.

CHAPTER 5 Tested cases

This chapter presents some applications of the `NEMESIGeo_R` library functionalities for three case studies [37]. The cores were chosen due to their different sizes and characteristics. It was also a good opportunity to practice with APOLLO3® code.

In order to verify the library's capabilities, one needs to test it by comparing the results arising from direct use of the library with some previously validated works. In this case, verification is not only restricted to the ability of the interface to reproduce the desired geometry. Coherency between benchmark transport calculations and calculations using the exported ALAMOS geometries is also required. To address this, the KAIST-like case presented during the CAMIVVER project is used as a benchmark to compare the results from a direct use of the reflector library.

To test both the library functionalities and APOLLO3® capabilities to deal with heavy reflector systems in a hexagonal lattice, the KZL6-mini core is studied, and results are compared with a TRIPOLI4® calculation. Here, the geometries produced by lattice and reflector libraries feed both TRIPOLI4® and APOLLO3® calculations, showing the versatility of these tools.

Finally, a crossover comparison between the industrial SCIENCE platform, APOLLO3®, and TRIPOLI4® Monte Carlo code is applied on a large-scale EPR core. The interest in studying this system is to apply the library to a real application. Therefore, this study brings valuable information on the sensitivity of various computational schemes, whether industrial or R&D.

5.1 The KAIST-like benchmark

The first case study to verify the library capabilities is the KAIST-like core. This core has been chosen due to its relatively small dimensions that allow to perform full core transport calculations without spatial homogenization in a reduced amount of time. Moreover, its reflector geometry has been defined to be representative of an EPR heavy reflector, and hence, it represents a good starting point for approaching EPR geometries.

The original KAIST core, described in [35], uses a water reflector (PWR-like). In the framework of the CAMIVVER project, a variation from the original geometry was proposed by substituting the water reflector with a heavy one [1]. This geometry was then used as a test case in the CAMIVVER project to validate MOC transport calculation in the presence of heavy reflectors and investigate reflector homogenization options for the traditional two-step computational schemes [34].

The results of a full core transport calculation presented by EDF during the final CAMIVVER workshop [8] are used here to perform a non-regression test of the library. The interest of this study is to verify that the usage of the `NEMESIGeo_R` library to reproduce the KAIST reflector geometry leads to results consistent with the ones resulting from the EDF study.

5.1.1 KAIST-like reactor

The KAIST-like reactor configuration is given in Figure 5.1. The core structure is taken from [35] while the reflector is one of the variants proposed in [1]. The reactor core has a one-eighth symmetry and is composed of MOX and UOX 17×17-pin fuel assemblies. The reflector is a heavy one, containing some water channels that are aligned and regularly distributed. An extended description of the materials and compositions used for the benchmark can be found in [34].

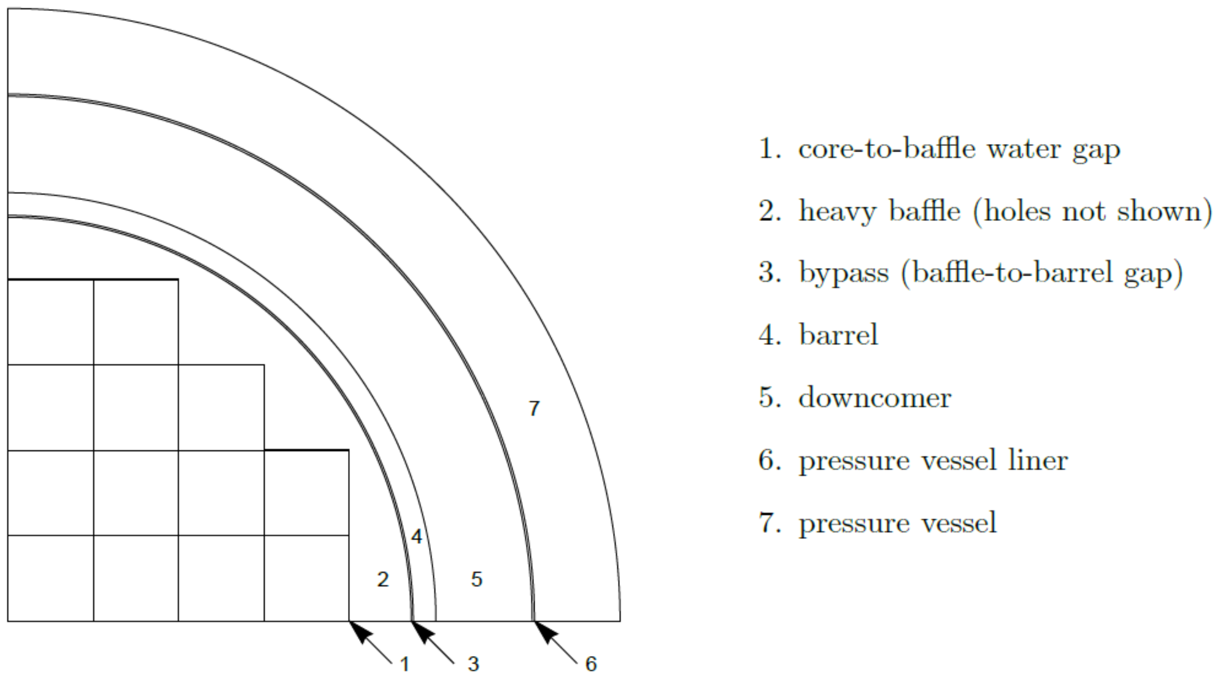

7
4
2
5
1
3
6
1. core-to-baffle water gap
2. heavy baffle (holes not shown)
3. bypass (baffle-to-barrel gap)
4. barrel
5. downcomer
6. pressure vessel liner
7. pressure vessel

Figure 5.1 Layout of the KAIST-like core with heavy reflector [1]

KAIST core structure

The core is composed of the following assembly types:

- Two homogeneous UOX assemblies, enriched at 2% and 3.3% respectively;

- An homogeneous UOX assembly with gadolinium;
- One heterogeneous MOX assembly;
- One homogeneous MOX assembly.

A detailed description of assemblies and their material composition is found in reference [34].

The KAIST-like heavy reflector

The chosen heavy reflector configuration is the first variant proposed in [1]. Its structure is given in 5.1. The pressure vessel made out of carbon steel has a thin stainless steel liner that separates it from the downcomer. The downcomer surrounds the barrel, separated from the heavy baffle by a slender water bypass. Both the baffle and barrel are made of stainless steel.

The baffle contains some water channels set in a regular array with the pitch in both directions equal to a fifth of the assembly pitch, as seen in Figure 5.2. Further details regarding the geometry are given in Table 5.1.

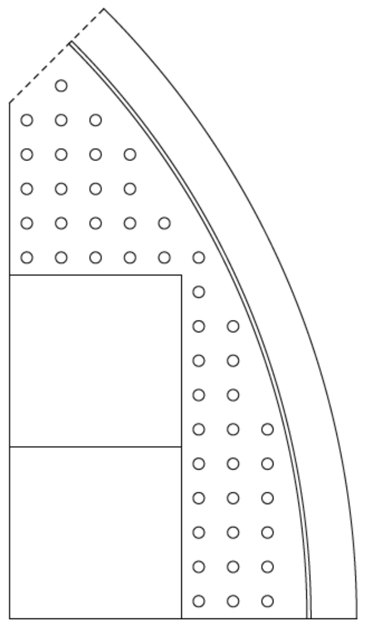


Figure 5.2 Details of the KAIST-like steel reflector, only the baffle and barrel are represented here [1]

Table 5.1 Dimensions of the pressure vessel internals for the KAIST-like heavy reflector [1]

	Dimensions in cm
assembly pitch	21.42
waterhole radius	0.700
bypass thickness	0.500
barrel inner radius	102.216
barrel outer radius	107.931
pressure vessel inner radius	132.793
pressure vessel outer radius	154.383

5.1.2 Methodology

Presentation of the EDF work

Dealing with heavy reflectors was one of the aspects treated during the CAMIVVER project. In order to investigate homogenization options for heavy reflectors, a full core reference transport calculation was performed by EDF and CEA on the KAIST-like reactor described previously using APOLLO3®. The results were compared to a Monte Carlo simulation to ensure their validity. The current work considers the full core transport calculation provided by EDF as a reference.

The geometry used for the calculation is obtained using APOLLO3® native geometries for assemblies, while the reflector is exported from a custom ALAMOS geometry. Some simplifications were performed to the original geometry to speed up calculation time and simplify the application of boundary conditions. As shown in Figure 5.3, the reflector is described only up to the downcomer having a rectangular shape, not a circular one.

The choice of this geometry stems from the assumption that the neutron flux reaching the downcomer is sufficiently low, causing the flux to disappear in that region, with the impact of the surrounding areas being negligible.

A relatively coarse refinement is applied to the reflector regions. Unlike the fuel pins, there is no necessity to finely refine the reflector regions due to their significantly lower flux. Consequently, the MOC convergence should be satisfactory, and introducing a finer refinement would result in an unnecessary computational time increase, without a notable enhancement in precision.

Output geometry is pin-by-pin for the core region, while the reflector is not homogenized, as shown in Figure 5.4. The self-shielding of the reflector is performed over a 2D simplified geometry where the reflector is represented by a 1D slice, alternating water and stainless

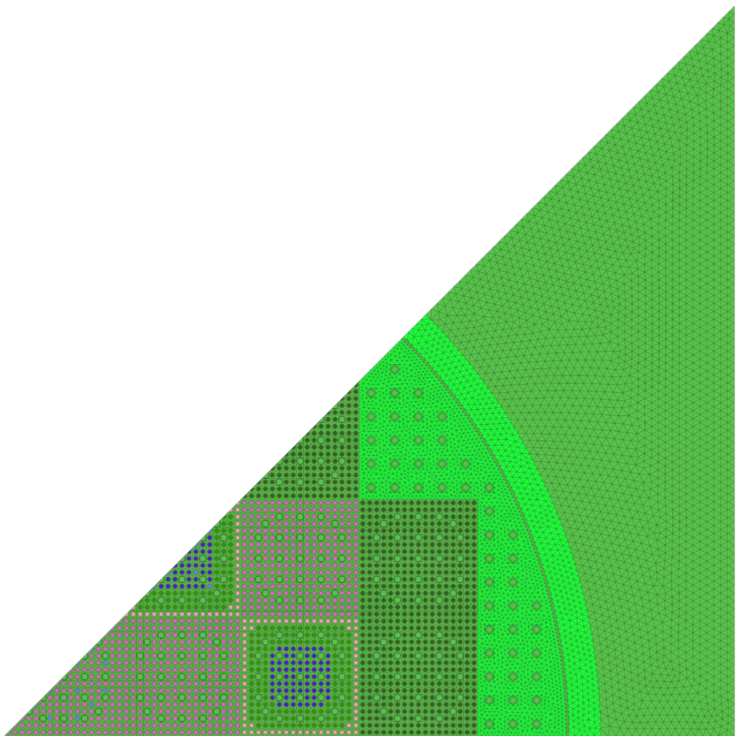


Figure 5.3 KAIST-like geometry for flux calculation [8]

steel according to the real geometry up to the downcomer, as shown in Figure 5.5.

Replicating the reflector's geometry

The reactor's geometry was replicated using two different approaches. The first, noted as NA-RL, uses a native core geometry and a reflector built using the `NEMESIGeo_R` library. The second approach, noted FA, employs a geometry fully assembled by ALAMOS using the `NEMESIGeo_C` and `NEMESIGeo_R` tools. A comparison between the original reference reflector geometry and the one generated using `NEMESIGeo_R` is given in Figure 5.6

Figure 5.6 shows that the KAIST-like reflector geometry can be easily generated using the library. The only differences between the two geometries are related to region refinements. The first approach is expected to show similar results to the reference case, with the reflector the only difference between the two geometries.

5.1.3 Presentation of the results

The EDF transport calculation shows a discrepancy of 200 pcm when compared to TRIPOLI4®. However, the primary focus of this study is to benchmark against the transport calculation

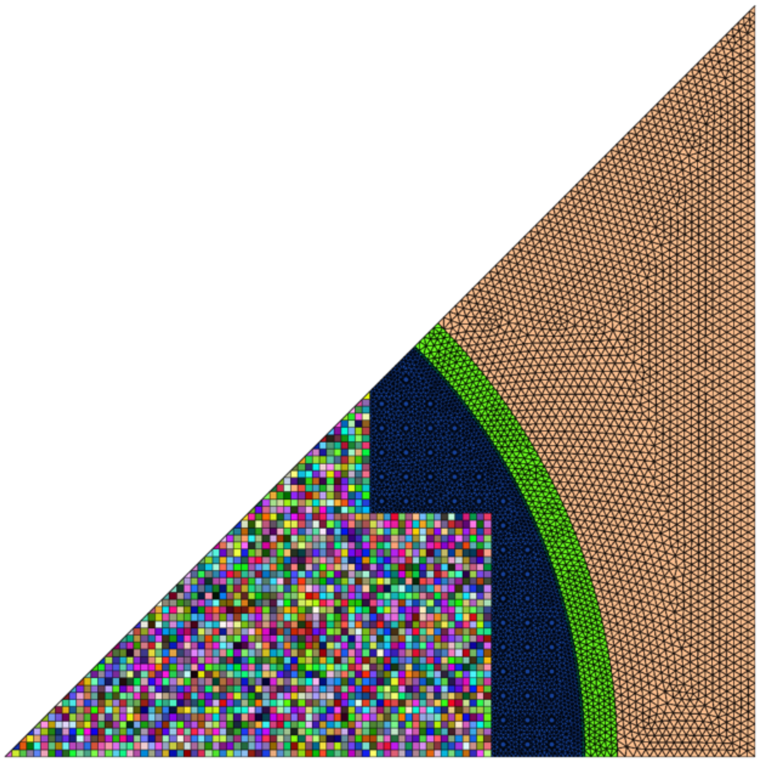


Figure 5.4 KAIST-like output geometry [8]

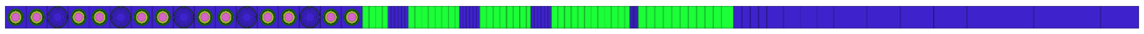


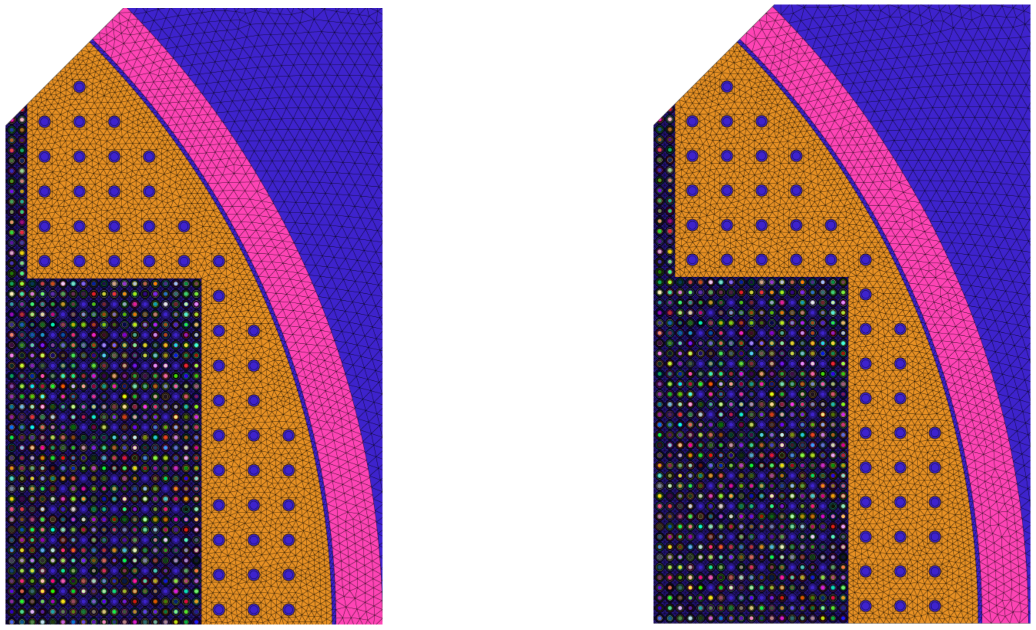
Figure 5.5 Geometry used for self-shielding calculations [8]

to ensure that the library's usage to generate the geometries does not introduce extra calculation biases. Therefore, we will refer to the transport calculation for comparisons in this context.

Table 5.2 presents results in terms of the effective multiplication factor. Distributions for power, flux, and reaction rates for fission and absorption have also been computed considering a two-group energy condensation and a pin-by-pin homogenization. Relative deviations with respect to the reference are given in Figures 5.7 and 5.8. Comparisons with the reference can only be done for the core regions since the reflector is not homogenized in the output geometry provided by EDF. The relative difference between reaction rates is computed using the following relation:

$$RD_i = \frac{RR_{\text{n, i}} - RR_{\text{ref, n, i}}}{RR_{\text{ref, n, i}}} \tag{5.1}$$

Where $RR_{\text{ref, n, i}}$ and $RR_{\text{n, i}}$ are reference and computed reaction rates respectively. Reaction rates are normalized with respect to the volume of cells and expressed in their adimensional


(a) Reflector geometry for the NA-RL study

(b) Original reflector geometry for the reference study

Figure 5.6 Comparison between reflector geometries. Only minor differences are found in the reflector meshes

form given by:

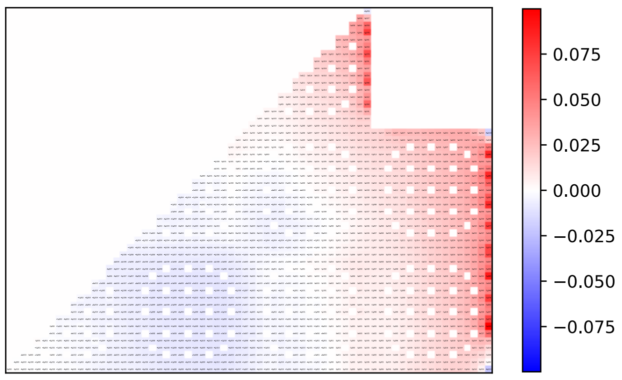
$$RR_{\text{n, i}} = \frac{RR_i}{V_i} \frac{\sum_i V_i}{\sum_i RR_i} \tag{5.2}$$

Table 5.2 Results of full core transport calculations for the KAIST-like reactor

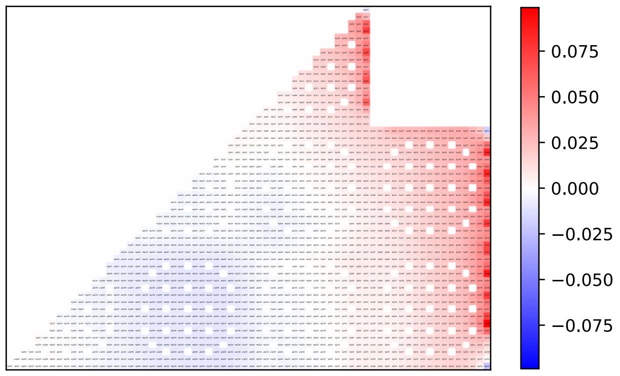
Model	Reference	NA - RL	FA
K_{eff}	1.13271	1.13267	1.13287
$\Delta\rho$ (pcm)	-	-3	+12

5.1.4 Analysis of the results

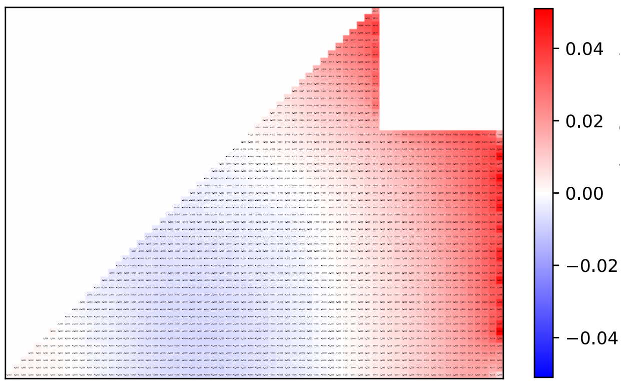
Results show very good agreement with the reference SHEM-MOC calculation provided by EDF. Considering the NA-RL model, only a 3 pcm difference in reactivity is found, and maximum reaction rate deviations from the reference do not exceed 0.1%. This outcome is in line with expectations, as the only difference between the two scenarios is the geometry of the reflector.



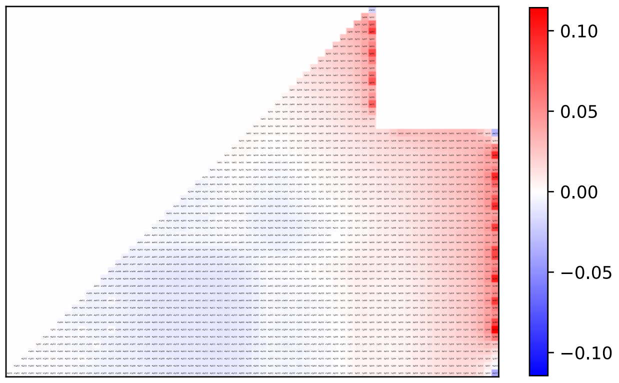
(a) Fission rate



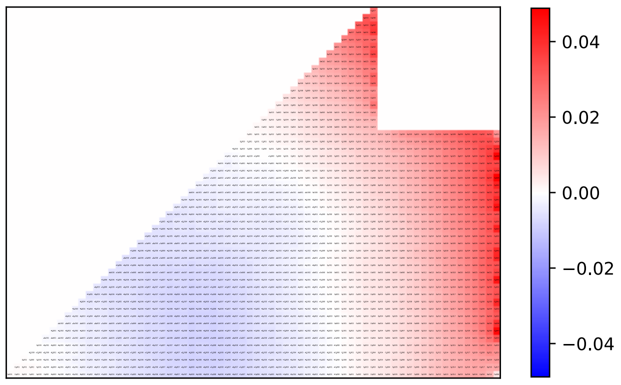
(b) Released energy



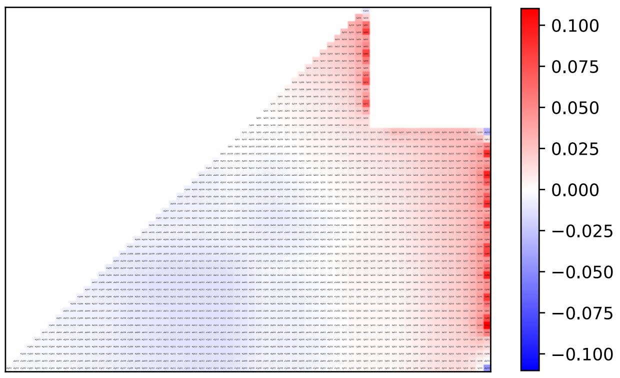
(c) Fast group absorption



(d) Thermal group absorption

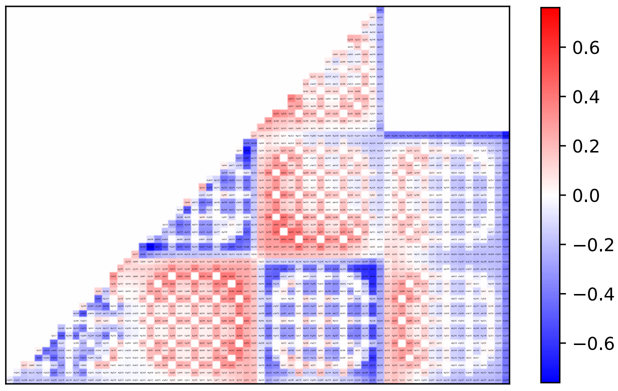


(e) Fast flux

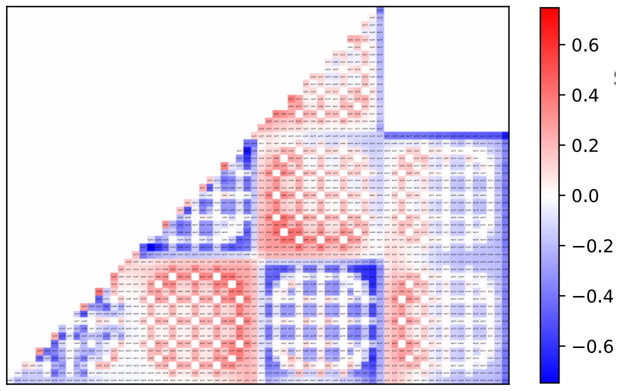


(f) Thermal flux

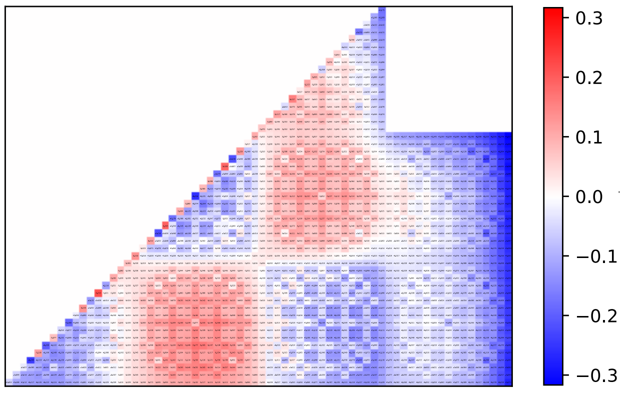
Figure 5.7 Relative differences with respect to the reference for the NA-RL case. Differences are expressed in percentages.



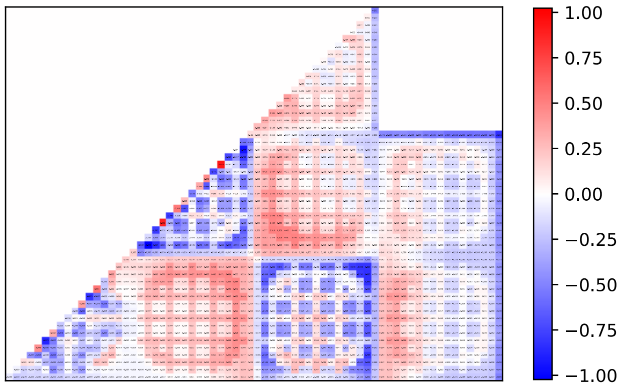
(a) Fission rate



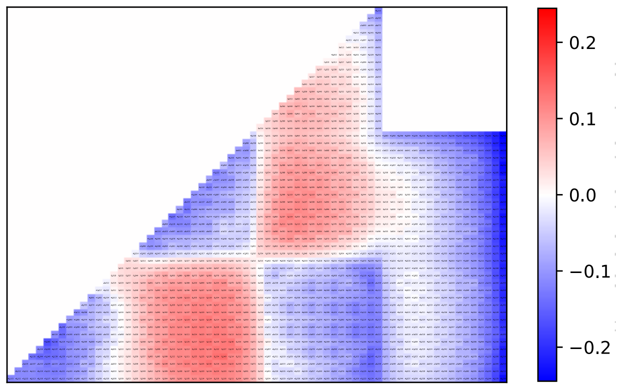
(b) Released energy



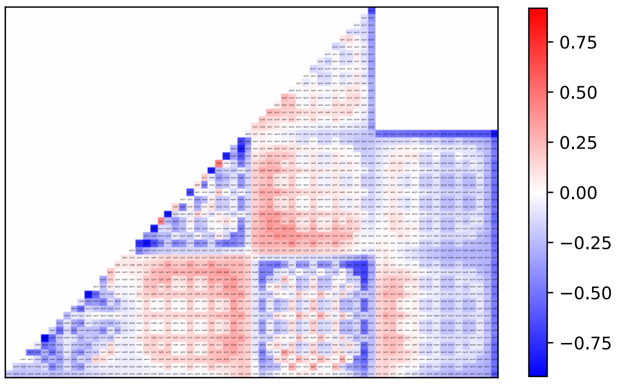
(c) Fast group absorption



(d) Thermal group absorption



(e) Fast flux



(f) Thermal flux

Figure 5.8 Relative difference with respect to the reference for the FA case. Differences are expressed in percentages.

The full ALAMOS model shows larger discrepancies. However, these are still very small, with reaction rates never larger than one percent and a difference in reactivity of 12 pcm. Differences in results are due to a diverse refinement applied to fuel assemblies. This application also shows the usefulness of lattice libraries in unlocking a simple approach to perform sensitivity analyses on the meshes.

It should be noted that this study does not bring any new information to the KAIST-like benchmark, which was already extensively studied during the CAMIVVER project [34]. Results align with expectations, proving that using the `NEMESIGeo_R` library to produce input geometry files leads to results consistent with similar calculations. Thanks to the newly produced libraries, a new, straightforward approach is available to produce input geometry files for APOLLO3® calculations. Libraries can also easily adjust specific parameters for sensitivity analysis purposes.

5.2 VVER minicore benchmark

To open the possibility of applying the available tools to simulate current operating VVER reactors, the library's capability of dealing with hexagonal cores is tested as well. A small VVER core benchmark was chosen among those presented in the CAMIVVER project [33,34], consisting of a seven-fuel assembly of the Kozloduy-6 (KZL6) reactor core with a one-sixth symmetry. In addition to an All-Rods-Out (ARO) configuration, this mini-core has been simulated using a configuration where control rods of the central fuel assembly are inserted.

This study also tests the capabilities of the TRIAGE module of APOLLO3®, which automatically produces a TRIPOLI4® input file using APOLLO3®. The computational meshes for the deterministic simulation and the reference stochastic simulations are shown in Figure 5.9, where both the geometries were produced by making use of the reflector library.

5.2.1 Results and discussion

For this test case, both the All-Rods-Out (ARO) and Assembly-Rods-In (ARI) configurations were tested and compared against a reference Monte Carlo simulation. For both cases, results from the APOLLO3® calculation show good agreement with the TRIPOLI4® reference results, in terms of K_{eff}, see Table 5.3, showing a difference in reactivity of -110 pcm and -118 pcm for the ARO and ARI configurations, respectively.

Figure 5.10 and Figure 5.11 represent absorption and fission rates for the thermal group for ARO and ARI configurations from the APOLLO3® calculation, showing the effect of control rod insertion on reaction rate distribution. Control rods cause a thermal absorption

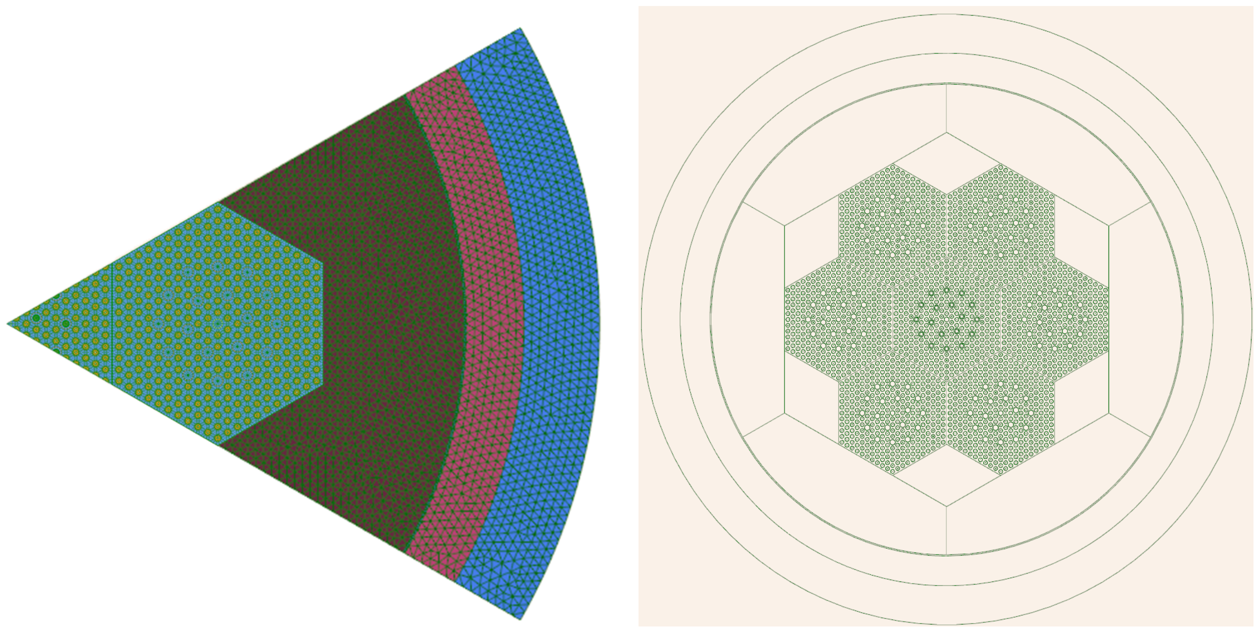


Figure 5.9 Meshes used for APOLLO3® (left) and TRIPOLI4® (right) calculations

peak in the central assembly (Figure 5.10b), leading to an important decrease in fission rates at the center of the core, as shown in Figure 5.11b. Accordingly, the ARI configuration shows its peak of released energy shifted towards the interface between the central assembly and the peripheral ones. Figure 5.12 represents the relative difference between APOLLO3® and TRIPOLI4® results for fission reaction rates for both the configurations. The ARO configuration shows a mean absolute error of 0.25% and the maximum error that is found is not larger than -1.3%. The ARI configuration shows instead deviations up to 2.49% with a mean absolute error of 0.35%. For both configurations, differences seem randomly distributed, alternating between positive and negative values. However, one can state that the results are in very good agreement. This can be due to the relatively high standard deviation associated to fission reaction rates scores in TRIPOLI4® which are up to 0.43% for ARO and 0.6% for ARI configurations. However, one can state that results are in good agreement.

Table 5.3 K_{eff} for ARO and ARI configurations of the KZL6-minicore (boron concentration: 600 ppm)

Configuration	APOLLO3®	TRIPOLI4® (Reference)
ARO	0.99937	1.00069 ($\sigma = 5$ pcm)
ARI	0.93364	0.93486 ($\sigma = 7$ pcm)

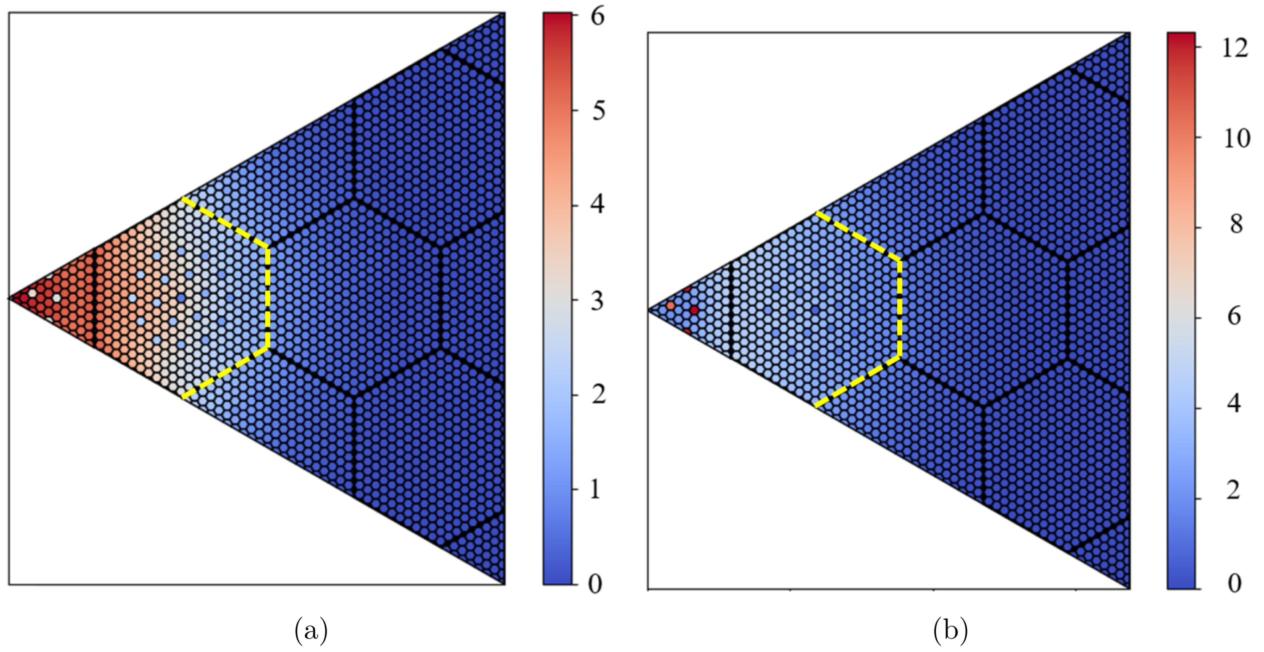


Figure 5.10 APOLLO3® thermal absorption rates in (a) ARO and (b) ARI configurations. The yellow dashed line marks the interface between the core and the reflector.

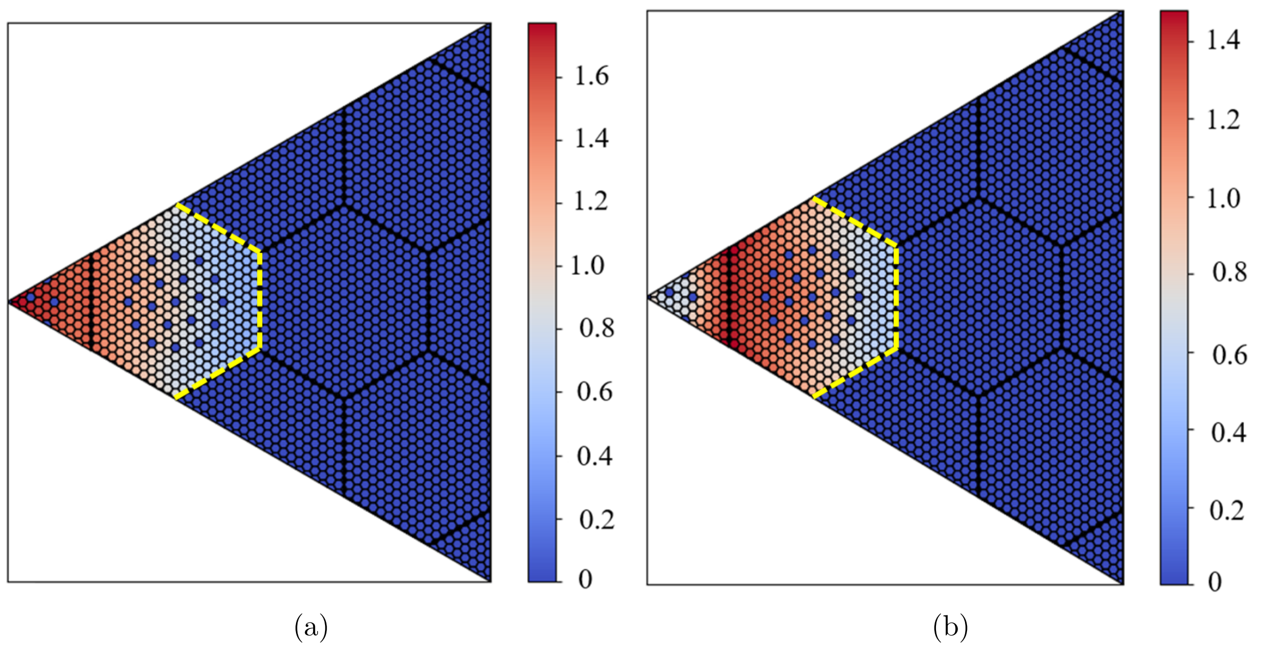


Figure 5.11 APOLLO3® thermal fission reaction rate for (a) ARO and (b) ARI configurations. The yellow dashed line marks the interface between the core and the reflector.

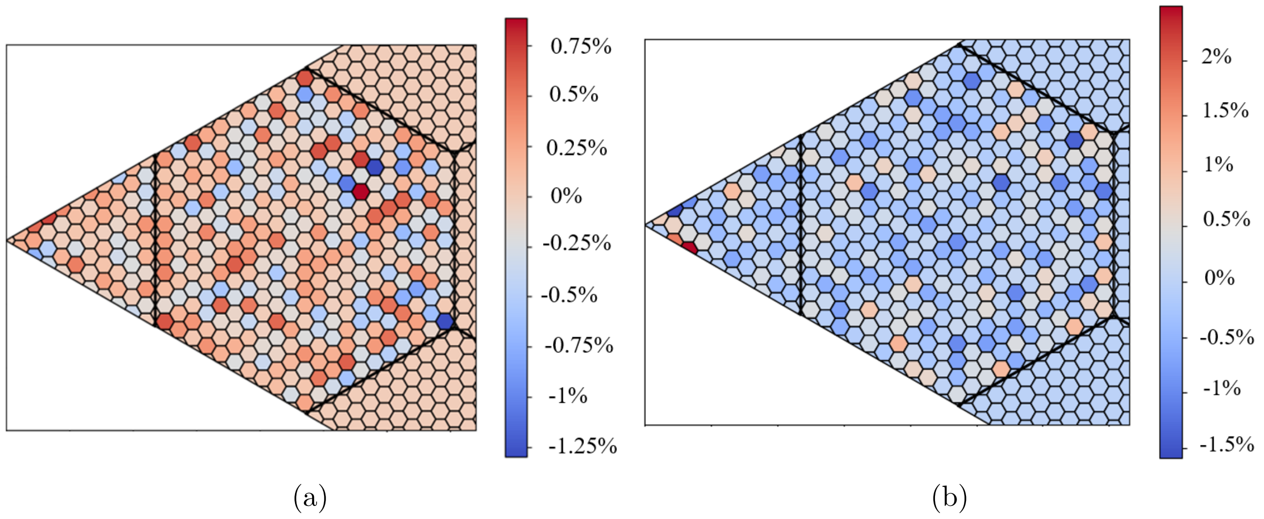


Figure 5.12 Relative difference for fission reaction rates with respect to the TRIPOLI4® reference for (a) ARO and (b) ARI configurations.

5.2.2 Towards VVER-1000 applications

To illustrate the capability of the `NEMESIGeo_R` library, a detailed representation of the VVER-1000 reflector geometry has been produced using the information available in [6]: the result is shown in Figure 5.13. The groove region is well represented, as shown in Figure 5.13. This example confirms the library's capabilities to support CAMIVVER follow-up activities.

5.3 EPR-like benchmark

The library flexibility is tested on a large-scale EPR-like reactor, which allows comparing different codes. The SCIENCE industrial platform is compared with the new generation APOLLO3® lattice code, which will be used to perform 2D full-core transport calculations. A TRIPOLI4® Monte Carlo reference is also provided, thanks to the TRIAGE module of APOLLO3®.

The SCIENCE platform uses JEF2.2 libraries, while only the JEFF3.1.1 library was available for TRIPOLI4® for the current work. Accordingly, a direct comparison between SCIENCE and TRIPOLI4® results would not be appropriate due to the significant differences between the two library versions. The two libraries are available in APOLLO3®. Accordingly, the APOLLO3® model is first validated against the TRIPOLI4® reference, using a 281 groups JEFF3.1.1 library. Then, the same APOLLO3® model, with a 99 groups JEF2.2 library is

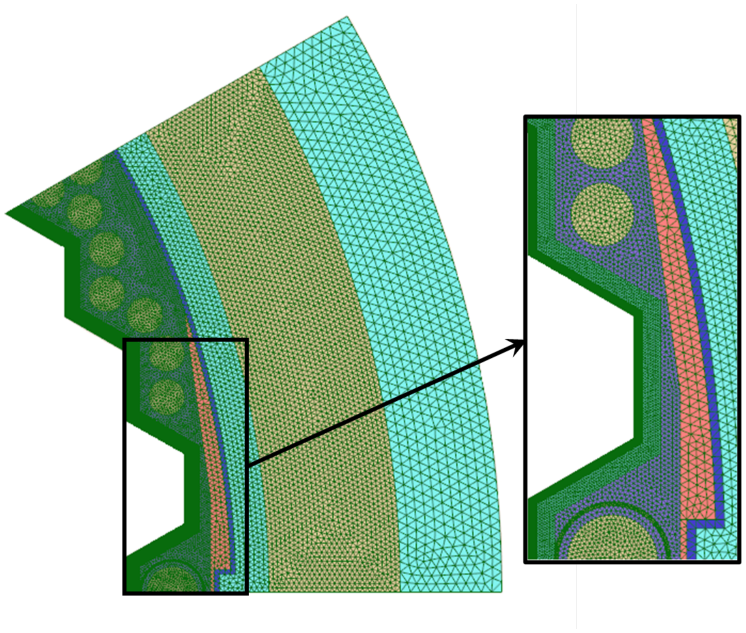


Figure 5.13 VVER-1000 reflector model and zoom on the groove region.

used as a reference to test SCIENCE accuracy, taking care to adjust the self-shielded groups for materials when changing the two libraries.

5.3.1 Setup of the geometry and composition of the APOLLO3® model

A full-scale case with a core composed of squared fuel assemblies surrounded by a heavy reflector has been considered. A 2D one-eighth model of an EPR-like core is proposed considering all its engineering components (water channels, tie rods and keys) in the reflector part, as shown in Figure 5.14. A spatial-dependent refinement shown in Figure 5.15 is applied in the reflector to ensure better accuracy in the regions close to the core. This spatial zone refinement is a feature implemented in the `NEMESIGeo_R` library to perform several sensitivity analyses. Figure 5.16a shows the refinement of the fuel pins and guide tubes applied for this test case, while water gaps are modeled according to Figure 5.16b.

The reactor is run at HZP conditions at 574K. The geometry and composition of assemblies is determined from a pre-existing study from the SCIENCE platform. The core is heterogeneous, with 5 different assembly types, according to Figure 5.14. Boron concentration in water is around 850 ppm and assembly grids dilution is neglected.

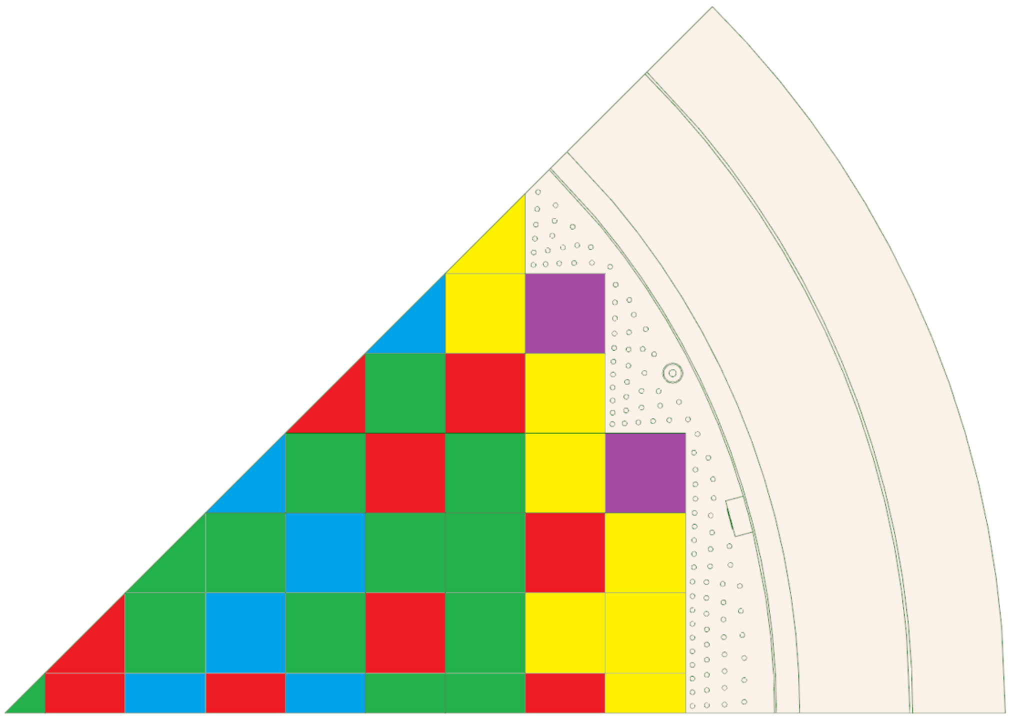


Figure 5.14 Sketch of the EPR core model. Different colors correspond to different fuel assemblies.

5.3.2 APOLLO3®/TRIPOLI4® comparison

Table 5.4 shows results in terms of K_{eff} obtained by an APOLLO3® 2D 1/8 full core model and TRIPOLI4® calculations. The discrepancy between APOLLO3® and TRIPOLI4® is 164 pcm. Reaction rates and the corresponding results from the APOLLO3® calculation for absorption are presented in Figures 5.17 and 5.18, while fission is presented in Figures 5.19 and 5.20. Results show that the highest peaks of Fission and Absorption are found close to the core's periphery, where assemblies contain higher enriched fuel. Comparisons are made for a quarter of assembly homogenization, showing that the absolute relative difference never exceeds 1.85%. Results for fission and absorption show similar discrepancies. More significant differences are found at the core-reflector interface, where the transport calculation underestimates the reaction rates. Future investigations oriented towards understanding these aspects and improving the calculation scheme will be possible thanks to the flexibility of the library developed. For instance, reflector self-shielding may be investigated to check its effects on the results, a more refined mesh could be applied at the core-reflector interface, or a more advanced MOC solver, such as the Linear Surface method [36], can be used to replace the Stepwise Constant scheme used in the present work.

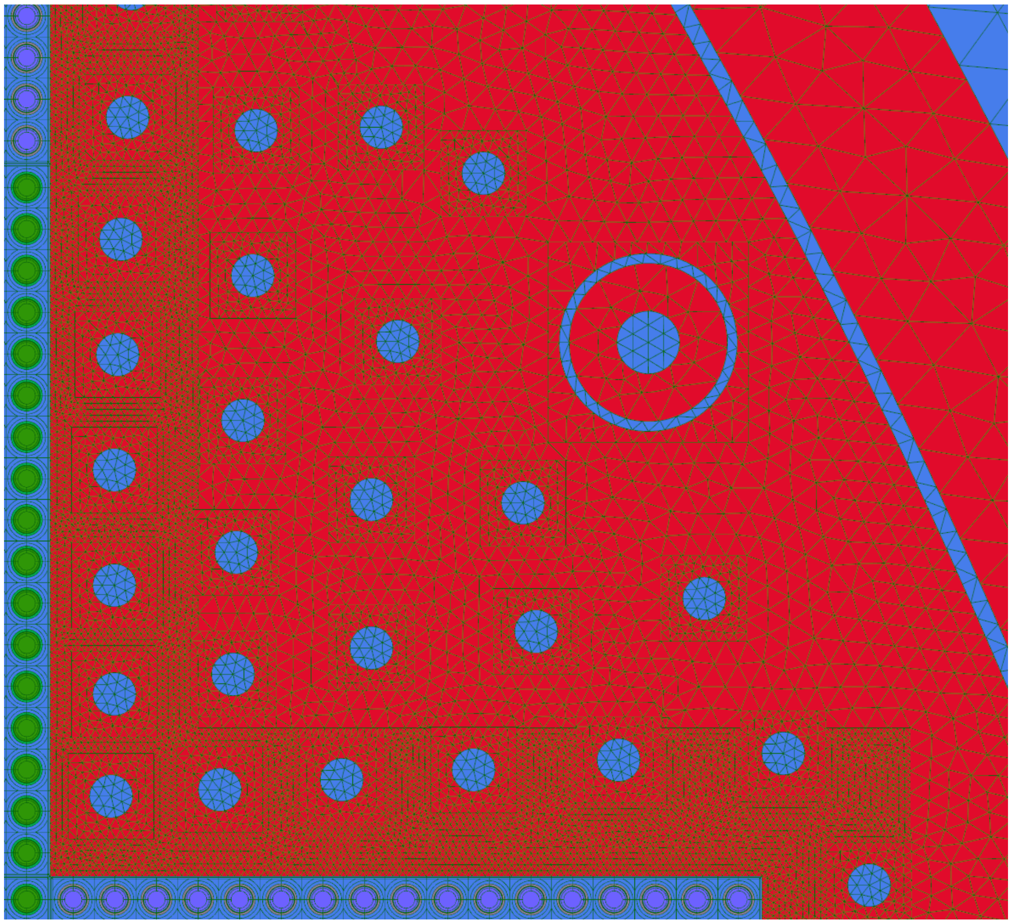
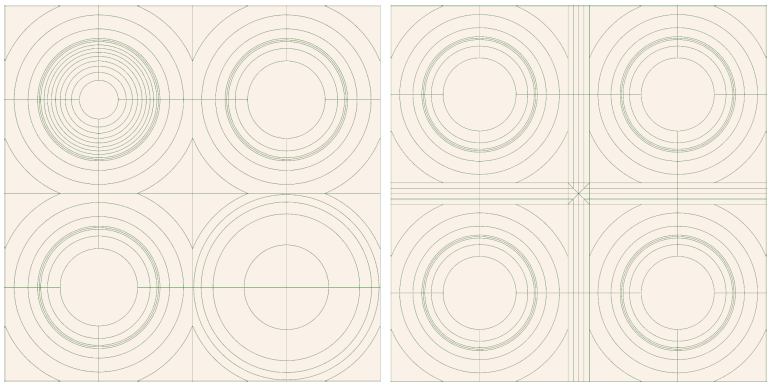


Figure 5.15 Close-up view of the refinement applied to the reflector



(a)

(b)

Figure 5.16 Refined meshed for (a) fuel pins and (b) water gaps

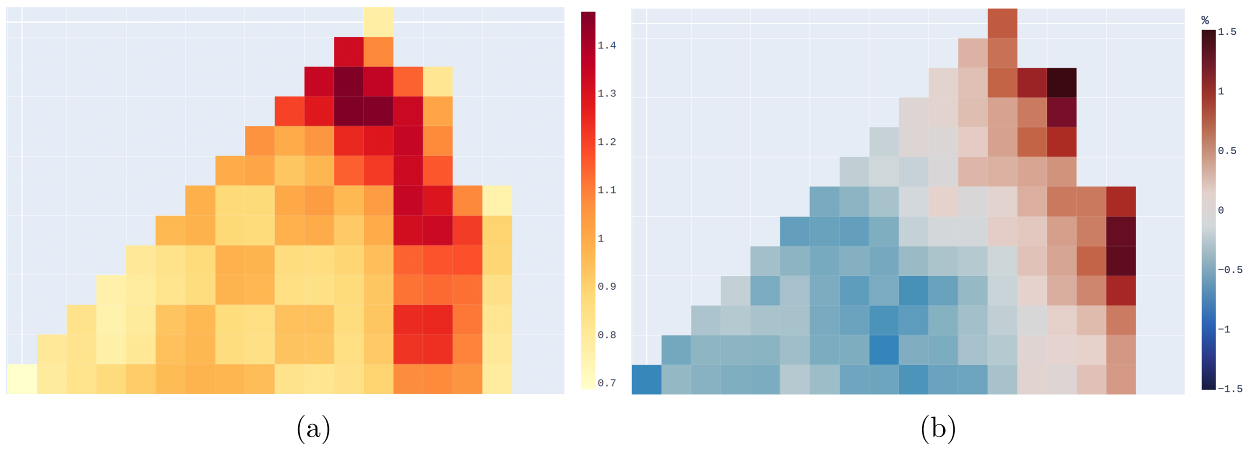


Figure 5.17 (a) APOLLO3® and (b) relative difference with respect to TRIPOLI4® results in absorption reaction rates for fast group.

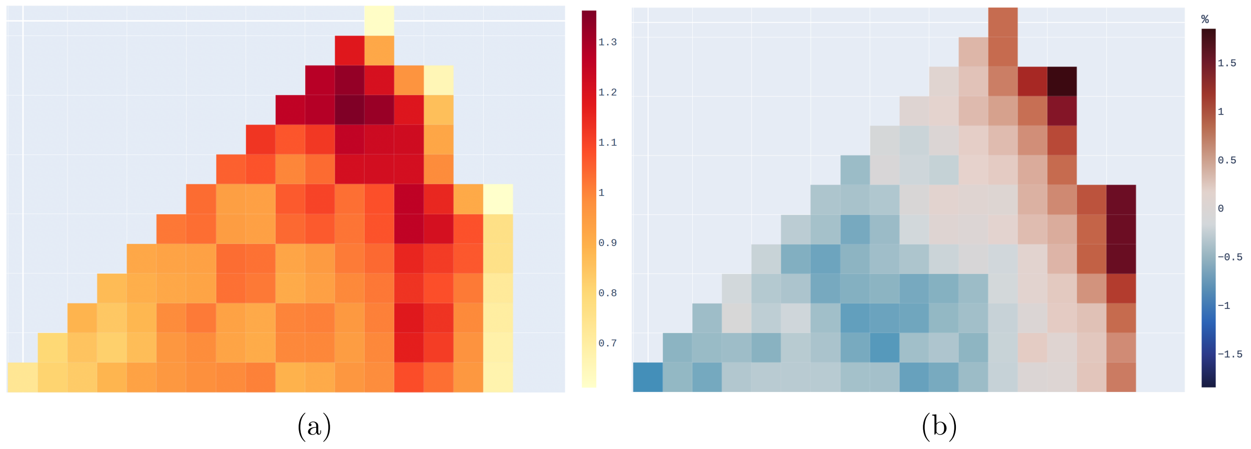
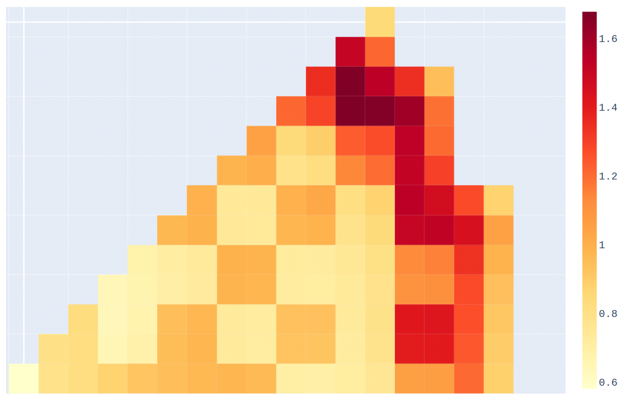
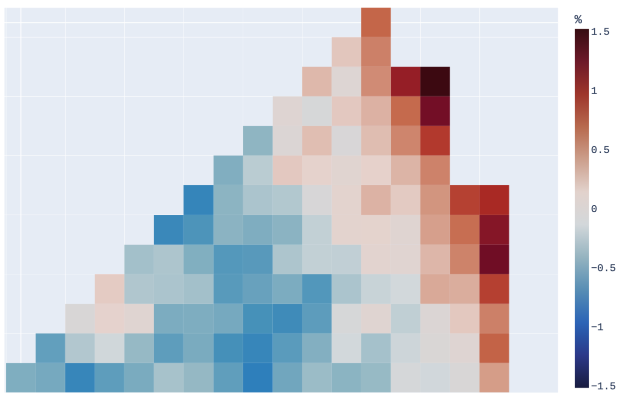


Figure 5.18 (a) APOLLO3® and (b) relative difference with respect to TRIPOLI4® results in absorption reaction rates for thermal group.

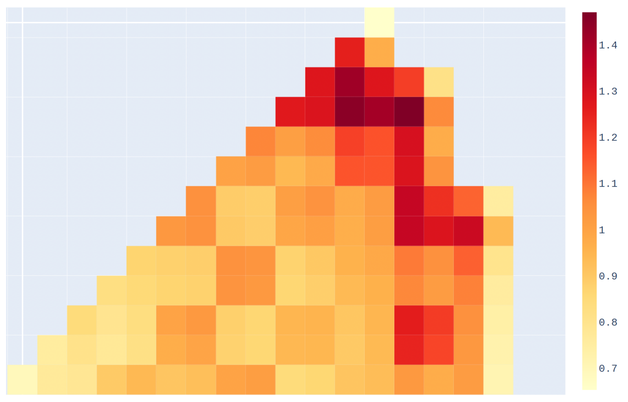


(a)

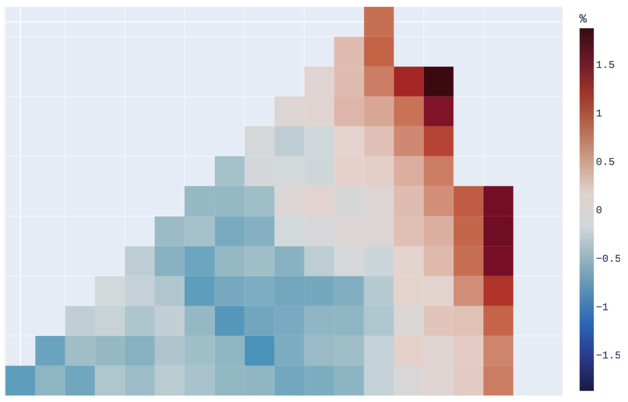


(b)

Figure 5.19 (a) APOLLO3® and (b) relative difference with respect to TRIPOLI4® for fast fission reaction rates



(a)



(b)

Figure 5.20 (a) APOLLO3® and (b) relative difference with respect to TRIPOLI4® for thermal fission reaction rates

Table 5.4 k_{eff} for the EPR-like configuration in HZP conditions (T = 574K, boron concentration: 850 ppm) using JEFF3.1.1.

Configuration	APOLLO3®	TRIPOLI4® (Reference)
EPR-like	0.99700	0.99863 ($\sigma = 5$ pcm)

5.3.3 SCIENCE/APOLLO3® comparison

After validation of the APOLLO3® 2D deterministic calculation with a reference Monte Carlo code, a similar calculation was performed using a 99 groups JEF2.2 cross section library to extend the comparison with the SCIENCE platform. SCIENCE includes a full-core solver named SMART based on the 3D nodal expansion method (NEM) with 2 energy groups. Here, the APOLLO3® model is used as a reference to test SCIENCE accuracy in modeling the EPR core. Table 5.5 displays the results in terms of the effective multiplication factor, exhibiting a 261 pcm difference between the two calculations.

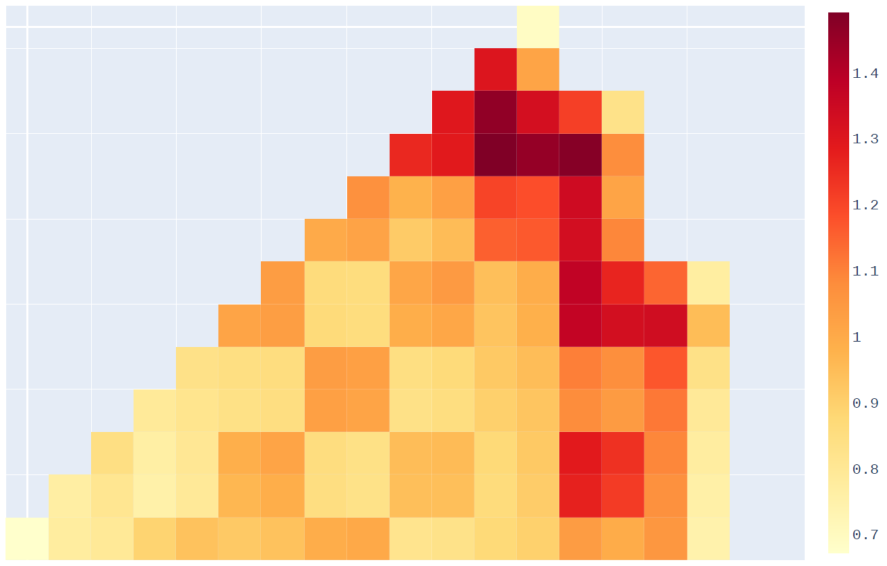
Table 5.5 k_{eff} for the EPR-like configuration in HZP conditions (T = 574K, boron concentration: 850 ppm) using JEF2.2

Configuration	SCIENCE®	APOLLO3® (Reference)
EPR-like	1.00288	1.00027

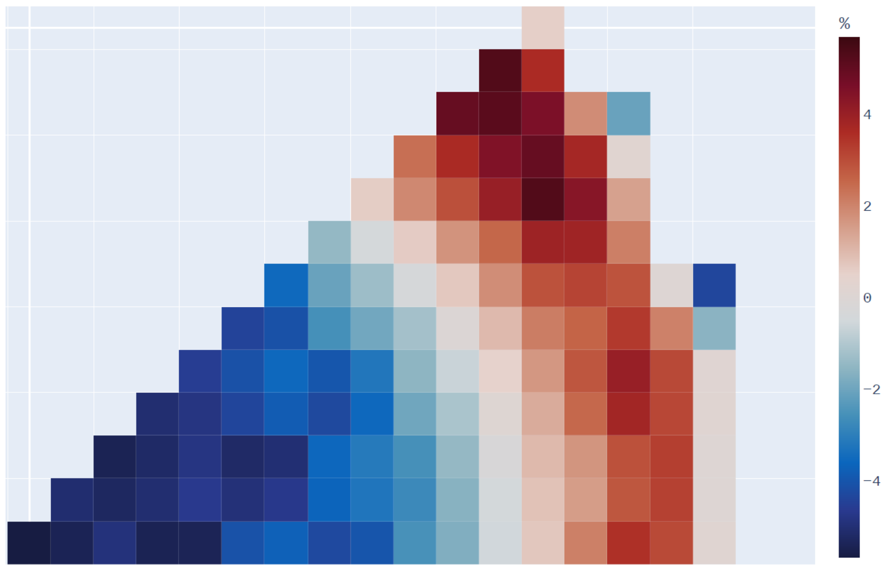
The two models exhibit some differences that may explain the discrepancies. The core solver in SCIENCE uses a 2 energy groups library for core calculations, while the APOLLO3® transport calculation is run using 99 energy groups. Reflector modeling is also different, since the APOLLO3® model uses a 2D reflector description, while the 1D Baff-Refl procedure (see Annex A.2) is used in SCIENCE. To investigate better these discrepancies, further studies of this kind should be done to test SCIENCE for other configurations.

Figure 5.21b shows the relative difference in energy release rate for a quarter of assembly homogenization. Results show that SCIENCE well predicts the energy release distribution around the core, being the errors always kept below 6% in the quarter of assembly configuration. Regions close to the reflector seem to be well modeled, showing deviations often close to zero. The only large deviations at the core-reflector interface are found at the corners, reaching up to -4.36%. These large deviations in the corners are in line with the expectations, since the 1D reflector modeling is not a good approximation of the corner regions. In particular, SCIENCE underpredicts the energy release in outgoing corners, while overpredicting it in the reentrant ones. Further studies should be performed to investigate this effect. Testing other slab configurations for the 1D reflector model in SCIENCE may be interesting to check if a more optimal choice of the reflector slab geometry and composition may reduce this effect.

Figure 5.23 displays the same comparison on a pin-by-bin basis, exhibiting the same pattern found in Figure 5.21. Results support the observations that have been made for the quarter of assembly configuration regarding the reflector modeling: the response of the reflector is well predicted in the regions far from the core corners, while energy release is underestimated in outgoing corners and overestimated in reentrant corners. Significant deviations are found in the pin regions corresponding to Gadolinium fuel cells. Here, SCIENCE largely overpredicts the energy release, with deviations reaching up to 20%. These results are in line with previous studies performed with SCIENCE.



(a)



(b)

Figure 5.21 APOLLO3® released energy rates (a) and its corresponding relative difference (b) with respect to SCIENCE in quarter of assembly configuration

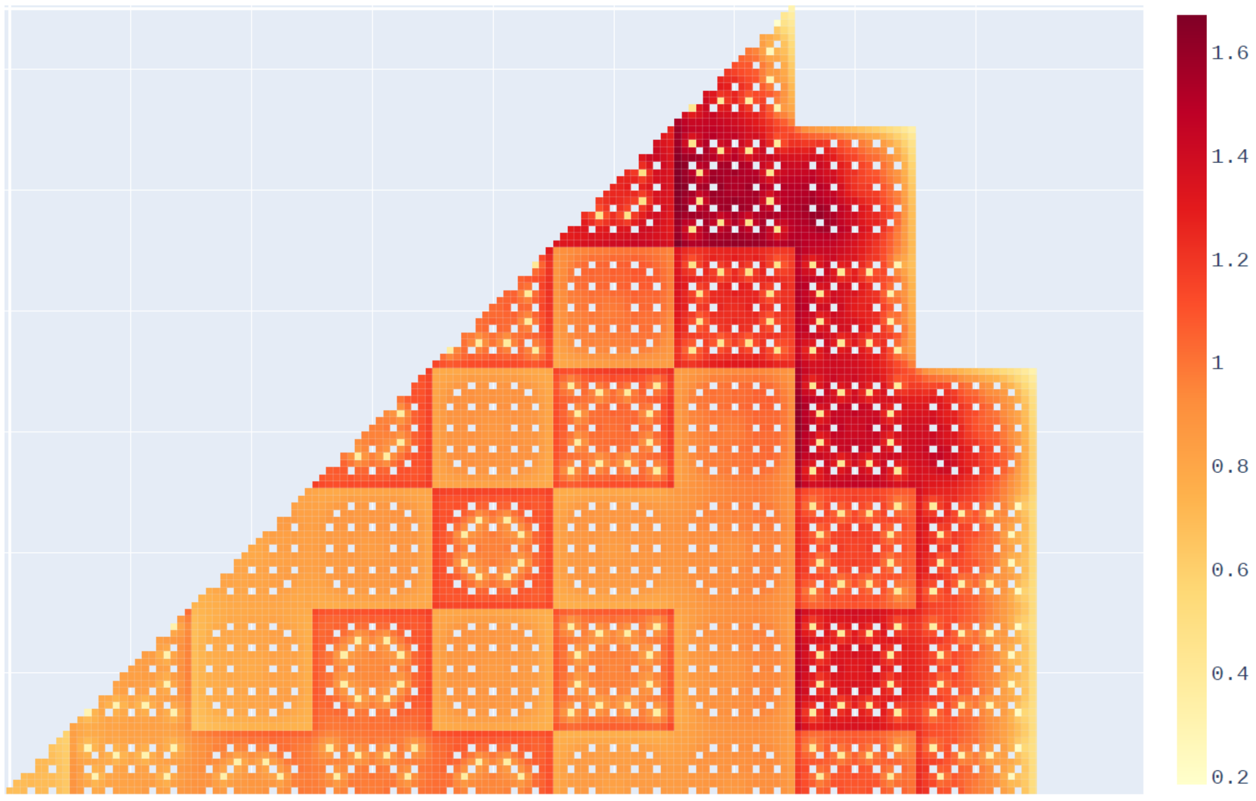


Figure 5.22 APOLLO3® released energy rates in pin-by-pin configuration

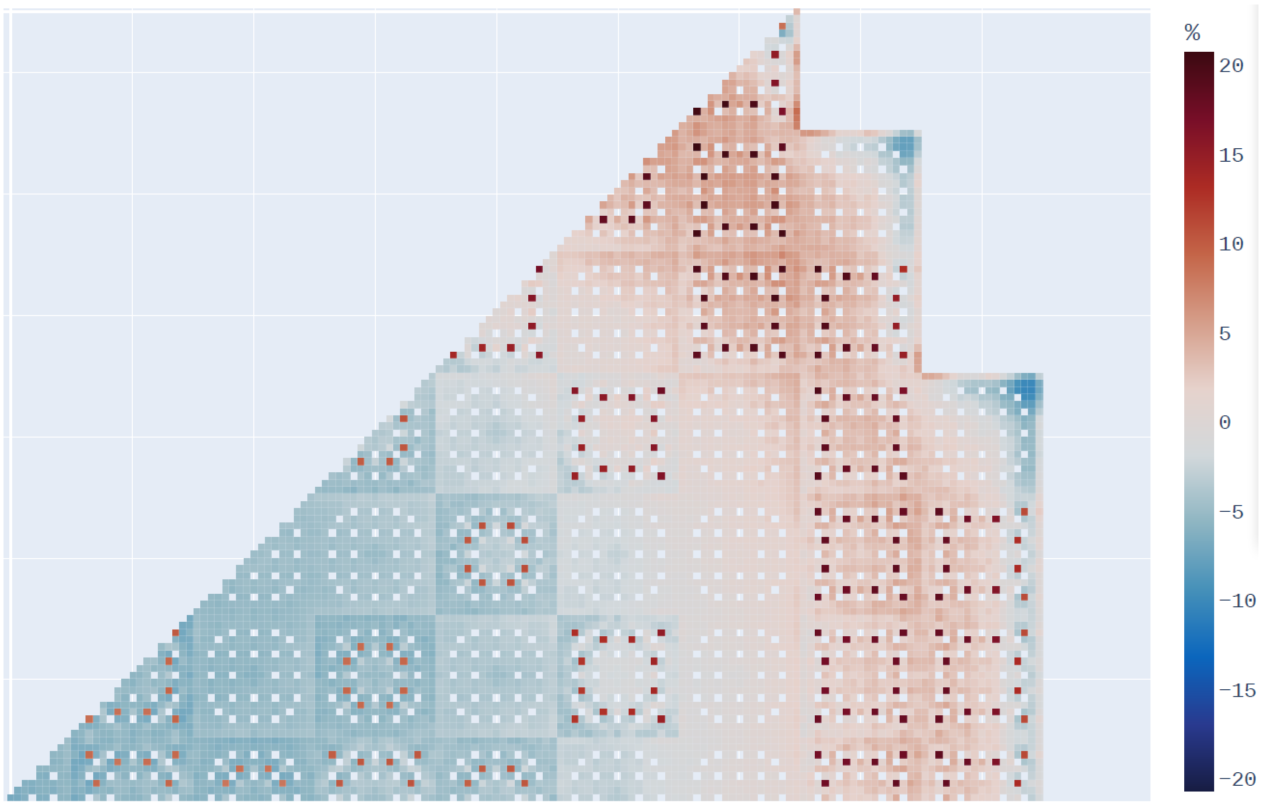


Figure 5.23 Relative difference with respect to SCIENCE for released energy rates in pin-by-pin configuration

CHAPTER 6 CONCLUSION

6.1 Summary of Works

The present work took a step further in developing NEMESI, a lattice multiparameter generator prototype developed within the H2020 CAMIVVER project for the industrialization of the APOLLO3® code. A Python library, called `NEMESIGeo_R`, based on the ALAMOS tool and oriented towards reflector modeling has been developed, unlocking the possibility for a rapid and straightforward generation of complex geometries for 2D lattice calculations. The library structure is presented by first providing an analysis of user needs based on a reflector design overview.

The library capabilities are investigated through three test cases [37], showing the potentiality of the library to produce detailed geometries for 2D reference deterministic calculations for both PWR and VVER configurations. A first testing of the library is provided, based on the KAIST-like core, showing that the `NEMESIGeo_R` library does not introduce extra calculation biases. VVER systems are investigated through a KZL6 minicore design from the CAMIVVER project, and results obtained via the APOLLO3® calculations show good agreement with respect to TRIPOLI4® reference Monte Carlo simulations. Potential applications to support future activities related to the CAMIVVER project follow-up are shown. A large scale application is also investigated using a pre-existing study of an EPR-like core in SCIENCE, representing a valuable testing of SCIENCE capabilities to deal with heavy reflector systems. A 2D full core transport model was designed in APOLLO3® and compared with a TRIPOLI4® reference and with the results from the SCIENCE platform. Comparisons with TRIPOLI4® are in good agreement, hence confirming the validity of the APOLLO3® model. Results from SCIENCE are in line with the expectations, the core-reflector region seems to be well modeled, with larger deviations found at the corners.

6.2 Limitations

The present work tested different R&D codes with complex configurations, discovering several bugs in the used sofwares (see Appendix C). The Python library developed is aimed to produce complex geometries and its applications allowed to figure out some bugs in ALAMOS linked in particular to the usage of circles. Applications on large geometries stressed the ALAMOS-APOLLO3® compatibility with configurations that are rarely employed in the conventional use of a lattice code, resulting in few bugs linked to the way APOLLO3®

interprets the ALAMOS geometries. Bugs were reported and rapidly corrected by CEA, which ensures the maintenance of these codes and continuously improves them, supporting the industrialization of APOLLO3® code and its robustness.

6.3 Future Research

The reflector library developed in the present work, along with the Framatome lattice libraries, opens the possibility to easily produce full core geometries for most reactors. Libraries are designed to be flexible; hence several models can be developed for the same reactor since performing mesh sensitivity analysis is made simpler. Accordingly, optimization of the geometry refinement parameters can be performed for all the selected test cases to improve results.

Considering the studied cases, the KZL6 model showed good agreement with the reference. However, a better statistic in the Monte Carlo calculation may allow to better understand the error distribution. The APOLLO3®/TRIPOLI4® comparison for the EPR-like core showed that larger deviations are found at the core-reflector interface. More investigations oriented towards understanding this aspect and improving the model have been depicted. Thank to the reflector library, reflector self-shielding effects may be investigated, or a more refined geometry can be applied at the core/reflector interface to ensure that flux gradients are well represented. One can also explore other MOC solver options available in APOLLO3®, such as the Linear Surface method.

Another interesting case study to be tackled is the VVER-1000 core, whose reflector is shown in the present work. It would be interesting to prove again APOLLO3® capabilities to deal with hexagonal cores, this time considering a large reactor with a complex reflector. Theoretical and experimental benchmarks are available in the literature, and it would be interesting to compare them with results from APOLLO3®, as proposed in the CAMIVVER follow-up project, currently under discussion at the European Commission.

This work gives an essential contribution to obtain 2D representation of reflector geometries with feeding assemblies. The next step is the generalization of Baff-Refl to 2D so as to produce discontinuity factors that are consistent with the MOC results.

The present work showed reflector library applications oriented towards 2D full-core reference transport calculations. However, the library can also deal with simplified reflector geometries, supporting future activities to improve reflector models, as the ones foreseen in the CAMIVVER follow-up activities.

REFERENCES

[1] I. Zmijarevic, "Two-dimensional PWR minicore model for transport calculations," in *CAMIVVER - revision 1, 2020.*

[2] A. Brighenti *et al.*, "Development of a multi-parameter library generator prototype for VVER and PWR applications based on APOLLO3®," in *Proc. Int. Conf. M&C2023, Niagara Falls, Ontario, Canada.*

[3] E. E. Lewis and W. F. Miller, *Computational methods of neutron transport*, 1 1984. [Online]. Available: https://www.osti.gov/biblio/5538794

[4] A. Hébert, *Applied Reactor Physics - Third Edition.* Presses internationales Polytechnique, 2020. [Online]. Available: http://www.presses-polytechnique.ca/en/applied-reactor-physics-third-edition

[5] "Table of nuclides," https://atom.kaeri.re.kr/nuchart/?zlv=2, accessed: 11/11/2023. [Online]. Available: https://atom.kaeri.re.kr/nuchart/?zlv=2

[6] Y. Bilodid, E. Fridman, and T. Lötsch, "X2 VVER-1000 benchmark revision: Fresh HZP core state and the reference Monte Carlo solution," *Annals of Nuclear Energy*, vol. 144, p. 107558, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306454920302565

[7] C. Sandrin, "Modélisation neutronique du réflecteur pour le calcul des cœurs des réacteurs nucléaires à eau pressurisée: Application à l'EPR," Rapport, CEA Saclay, Direction de l'Énergie Nucléaire, Direction Déléguée aux Activités Nucléaires, Département de Modélisation des Systèmes et Structures, Service d'Études des Réacteurs et de Mathématiques Appliquées, CEA/Saclay 91191 Gif-sur-Yvette Cedex, France, 2010, iSSN 0429-3460.

[8] E.-Y. Garcia-Cervantes and M. Tiberga, "Advanced features of APOLLO3® lattice code in the framework of the CAMIVVER project," 7 2023, workshop CAMIVVER, Karlsruhe. [Online]. Available: https://zenodo.org/records/8183023

[9] A. Hébert, "A review of nodal and SPH equivalence techniques applied to reflector models in PWRs," Presentation date: 2022/09/21, September 2022. [Online]. Available: http://merlin.polymtl.ca/downloads/reflector_model_sept2022.pdf

[10] Autorité de Sûreté Nucléaire, "Guide ASN 28: Qualification des outils de calcul scientifique utilisés dans la démonstration de sûreté nucléaire - première barrière," Online, 2017. [Online]. Available: https://www.asn.fr/l-asn-reglemente/guides-de-l-asn/

[11] P. Girieu *et al.*, "SCIENCE Version 2: The most recent capabilities of the Framatome 3D nuclear code package," FRAMATOME Nuclear Operations, Technical Report Tour FRAMATOME Cedex 16 92084 PARIS La Défense, 1999. [Online]. Available: https://inis.iaea.org/collection/NCLCollectionStore/_Public/33/010/33010761.pdf

[12] E. Martinolli *et al.*, "APOLLO2-A - Areva's new generation lattice physics code: Methodology and validation," in *PHYSOR 2010, Pittsburg, USA*, vol. 2.

[13] D. Tomatis *et al.*, "Overview of SERMA's graphical user interfaces for lattice transport calculations," *Energies*, vol. 15, 02 2022.

[14] P. Mosca *et al.*, "APOLLO3®: Overview of the new code capabilities for reactor physics analysis," in *Proc. Int. Conf. M&C2023*, August 2023, pp. 13–17.

[15] D. Verrier *et al.*, "Codes and methods improvements for VVER comprehensive safety assessment: The CAMIVVER H2020 project," in *ICONE28 - 28th International Conference on Nuclear Engineering*, 08 2021.

[16] E. Brun *et al.*, "TRIPOLI-4®, CEA, EDF and AREVA reference Monte Carlo code," *Annals of Nuclear Energy*, vol. 82, pp. 151–160, 2015.

[17] Salome. [Online]. Available: https://salome-platform.org

[18] D. Schneider *et al.*, "APOLLO3®: CEA/DEN deterministic multi-purpose code for reactor physics analysis," in *PHYSOR 2016 – Unifying Theory and Experiments in the 21st Century, May 2016, Sun Valley, United States.*

[19] J.-F. Vidal *et al.*, "Qualification of JEFF3.1.1 library for high conversion reactor calculations using the ERASME/R experiment," in *PHYSOR 2012, Knoxville, USA*.

[20] Nuclear Energy Agency (NEA), "The JEF-2.2 nuclear data library," Paris, 2000.

[21] J. J. Duderstadt and L. J. Hamilton, *Nuclear reactor analysis.* Wiley New York, 1976.

[22] A. Fortin and A. Garon, *Les éléments finis: de la théorie à la pratique.* Université Laval, 2020. [Online]. Available: https://giref.ulaval.ca/afortin/elements_finis.pdf

[23] R. Le Tellier, “Développement de la méthode des caractéristiques pour le calcul de réseau,” Ph.D. dissertation, Polytechnique Montréal, 2006.

[24] M. Livolant and F. Jeanpierre, “Autoprotection des résonances dans les réacteurs nucléaires - application aux isotopes lourds,” *CEA-R-4533, Commissariat à l'Energie Atomique, France*, 1974.

[25] F. Simon, “Validation de la méthode SPM pour l'autoprotection des résonances,” Master's thesis, Polytechnique Montréal, 2011.

[26] R. Macfarlane *et al.*, “The NJOY Nuclear Data Processing System, version 2016,” United States, Tech. Rep. LA-UR–17-20093, Jan 2017.

[27] R. Stammler and M. Abbate, *Methods of Steady-State Reactor Physics in Nuclear Design.* United Kingdom: Academic Press, 1983.

[28] A. Hébert and G. Marleau, “Generalization of the Stamm'ler Method for the Self-Shielding of Resonant Isotopes in Arbitrary Geometries,” *Nucl. Sci. Eng.*, vol. 108, p. 230, 1991.

[29] M. Coste-Delclaux, “Modélisation du phénomène d'autoprotection dans le code de transport multigroupe APOLLO2,” Ph.D. dissertation, CEA, Gif-Sur-Yvette, France, 2006.

[30] V. Salino, “Incertitudes et ajustements de données nucléaires au moyen de méthodes déterministes, probabilistes et de mesures effectuées sur des réacteurs à eau sous pression,” Ph.D. dissertation, Polytechnique Montréal, May 2022.

[31] K. Smith, “Assembly homogenization techniques for light water reactor analysis,” *Progress in Nuclear Energy*, vol. 17, no. 3, pp. 303–335, 1986. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0149197086900351

[32] A. Previti *et al.*, “Towards a systematic requirement-based approach to build a neutronics study platform,” *Nuclear Science and Engineering*, vol. 197, pp. 2459–2483, 2023.

[33] J. Blanco and B. Calgaro, “D5.1 - description of the core reference test cases – part 1,” CAMIVVER, Tech. Rep., 2023, accessed: 2023-09-20. [Online]. Available: http://camivver-h2020.eu/src/assets/doc/D5-1.pdf

[34] A. Willien and B. Vezzoni, “D4.3 – definitions of tests cases for the verification phases of the multi-parametric library generator,” CAMIVVER, Tech. Rep., 2023, accessed: 2023-09-20. [Online]. Available: http://camivver-h2020.eu/src/assets/doc/D4-3.pdf

[35] N. Z. Cho, "Benchmark problem 1a: MOX fuel-loaded small PWR core (mox fuel with zoning)," 2020. [Online]. Available: http://nurapt.kaist.ac.kr/benchmark

[36] M. Tiberga and S. Santandrea, "A novel high-order surface characteristics scheme for the neutron transport equation in 2D unstructured meshes," *Nuclear Science and Engineering*, pp. 1–45, 2023. [Online]. Available: https://doi.org/10.1080/00295639.2023.2214488

[37] F. Inzirillo *et al.*, "Towards 2D reference deterministic calculations in support of industrial model verification and advanced reflector modeling setup," in *PHYSOR 2024 [Submitted], San Francisco, USA*.

[38] T. Clerc *et al.*, "An advanced computational scheme for the optimization of 2D radial reflector calculations in pressurized water reactors," *Nuclear Engineering and Design*, vol. 273, pp. 560–575, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029549314002052

[39] S. Machach, "Étude des techniques d'équivalence nodale appliquées aux modèles de réflecteurs dans les réacteurs à eau pressurisée," Master's thesis, Polytechnique Montréal, 2022.

APPENDIX A REFLECTOR MODELS

A.1 Lefebvre-Lebigot model

The model calculates the neutronic parameters of a homogeneous reflector, equivalent to the heterogeneous one as shown in Figure A.1. The heterogeneous system is solved using the S_n method and is used as a reference to determine the two-group neutronic parameters for the reflector, such as diffusion coefficients and cross-sections [30].

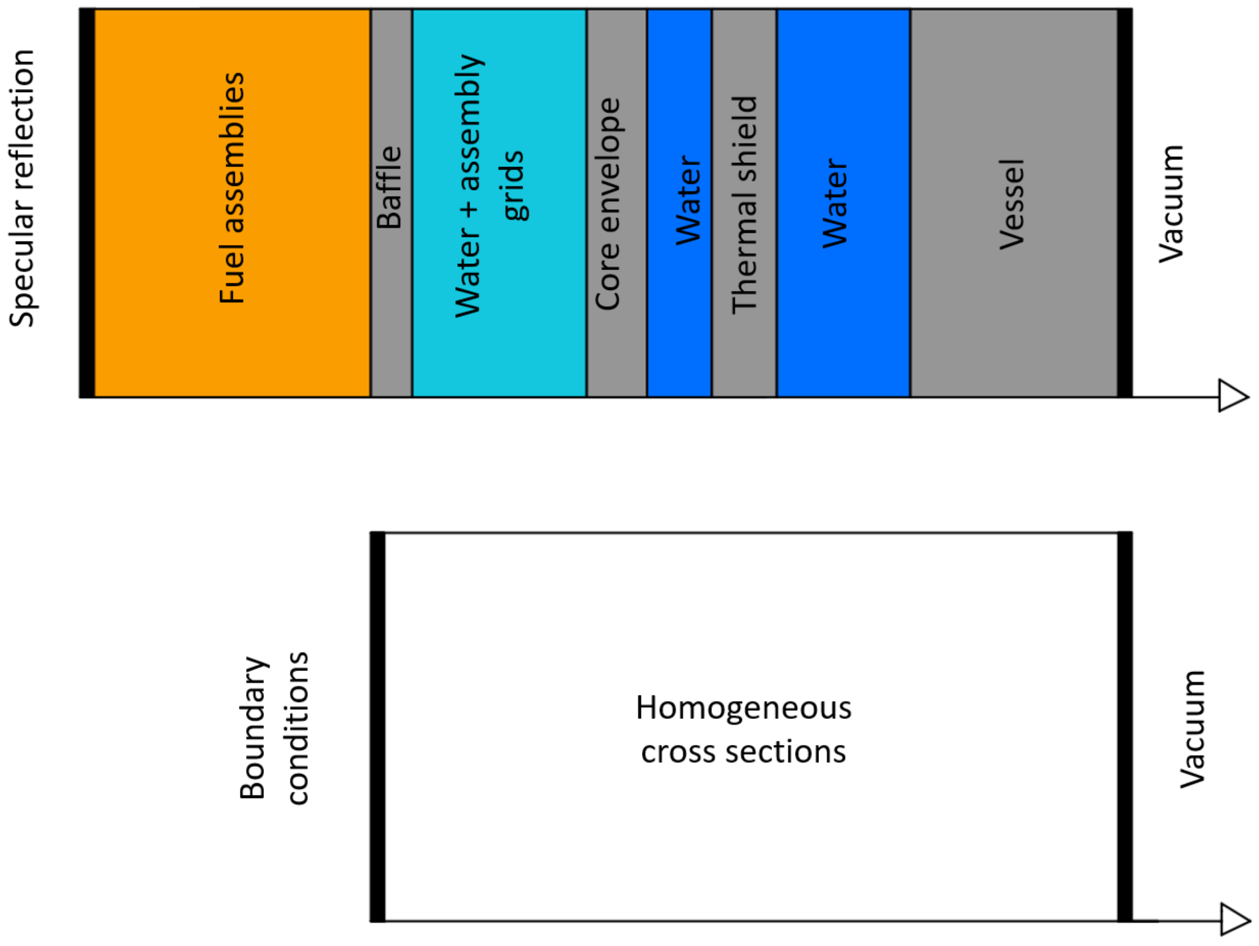


Figure A.1 Top: A one-dimensional cross-section depicting a core's edge. Bottom: A homogeneous cross-section to be provided with equivalent effective sections.

The homogeneous reflector is described by the two-group diffusion equation for a non-multiplying medium, without upscattering.

$$-D_1\Delta\phi_1(x) + (\Sigma_{a1} + \Sigma_{1\to s2})\phi_1(x) = 0 \tag{A.1}$$

$$-D_2\Delta\phi_2(x) + \Sigma_{a2}\phi_2(x) = \Sigma_{1\to 2}\phi_1(x) \tag{A.2}$$

Where D_1 and D_2 are the diffusion coefficients of group 1 and 2 respectively. Analytical solutions are available for both the current and flux as functions of the reflector's neutronic parameters D_1, D_2, Σ_{a1}, Σ_{a2}, $\Sigma_{1\to 2}$ and boundary conditions $\phi_1(0)$ and $\phi_2(0)$ at the interface with the core.

A reflector equivalence model is aimed at preserving some characteristics of current and flux at the interface between the reflector and the core. Accordingly, only the behavior at $x = 0$ is of interest. Quantities of interest to be preserved are for sure the currents J_1 and J_2 at $x = 0$, which can be expressed, by omitting the indices indicating $x = 0$ to simplify the notation, as:

$$\frac{J_1}{\phi_1} = \sqrt{D_1(\Sigma_{a1} + \Sigma_{1\to 2})} \tag{A.3}$$

$$\frac{J_2}{\phi_1} = \frac{\phi_2}{\phi_1}\sqrt{D_2\Sigma_{a2}} - \frac{\Sigma_{1\to 2}\sqrt{D_1 D_2}}{\sqrt{\Sigma_{a2}D_1} + \sqrt{(\Sigma_{a1} + \Sigma_{1\to 2})D_2}} \tag{A.4}$$

The desired homogenized reflector must be compatible with any kind of fuel, at any level of enrichment, burnup or temperature. Accordingly, several numerical studies are performed at different conditions for the fuel, the most pertinent results are described below.

Considering the fast group, it is found that the ratio J_1/ϕ_1 only depends on reflector geometry and composition. The ratio is almost independent on fuel characteristics, and is renamed as:

$$R_1 = \frac{J_1}{\phi_1} \tag{A.5}$$

For the thermal group, it is found that the ratio J_2/ϕ_1 depends on fuel parameters. By computing the ratio for different fuels, a linear dependency with ϕ_2/ϕ_1 is found. Hence, two more unknowns R_2 and R_3 are defined as the coefficients of the linear relation:

$$\frac{J_2}{\phi_1} = R_2\frac{\phi_2}{\phi_1} + R_3 \tag{A.6}$$

The three coefficients are all determined from the reference S_n calculation performed on the heterogeneous geometry. By using equations A.3 and A.4, these three coefficients are analytically linked to the cross sections of the equivalent reflector:

$$R_1 = \sqrt{D_1(\Sigma_{a1} + \Sigma_{1\to 2})} \tag{A.7}$$

$$R_2 = \sqrt{D_2 \Sigma_{a2}} \tag{A.8}$$

$$R_3 = \frac{\Sigma_{1\to 2}\sqrt{D_1 D_2}}{\sqrt{\Sigma_{a2} D_1} + \sqrt{(\Sigma_{a1} + \Sigma_{1\to 2}) D_2}} \tag{A.9}$$

Only two variables remain to be determined to acquire the five neutronic properties of the equivalent reflector (D_1, D_2, Σ_{a1}, Σ_{a2} and $\Sigma_{1\to 2}$). For better numerical convergence in diffusion codes, Lefebvre and Lebigot recommend setting the diffusion coefficients of the reflector to match (or closely resemble) those of its neighboring assemblies.

Accordingly, cross-sections for the equivalent reflector are given by:

$$\Sigma_{a2} = \frac{(R_2)^2}{D_2} \tag{A.10}$$

$$\Sigma_{1\to 2} = R_3 \left(\frac{R_1}{D_1} + \sqrt{\frac{\Sigma_{a2}}{D_2}} \right) \tag{A.11}$$

$$\Sigma_{a1} = \frac{(R_1)^2}{D_1} - \Sigma_{1\to 2} \tag{A.12}$$

This model turn out to be very effective in computing reflector properties and is currently used in industrial schemes at EDF. However, it is only applicable to two-group diffusion theory and to 1D reflector models [38]. Another limitation comes from imposing the diffusion coefficient to be equal to arbitrary values. This limitation is not present in Baff-Refl model [9, 39], developed recently at Framatome and implemented in the SCIENCE platform.

A.2 Baff-Refl procedure

The Baff-Refl procedure allows for equivalence between a reference calculation made on a 1D heterogeneous geometry and a calculation on a homogenized slab resolved using a nodal expansion method as shown in Figure A.2. The nodal expansion method improves the accuracy of finite difference schemes by developing the flux and currents with polynomial functions.

The procedure is based on a S_n reference calculation performed over the heterogeneous geometry. Two feeding assemblies are generally used, and their cross sections are recovered from

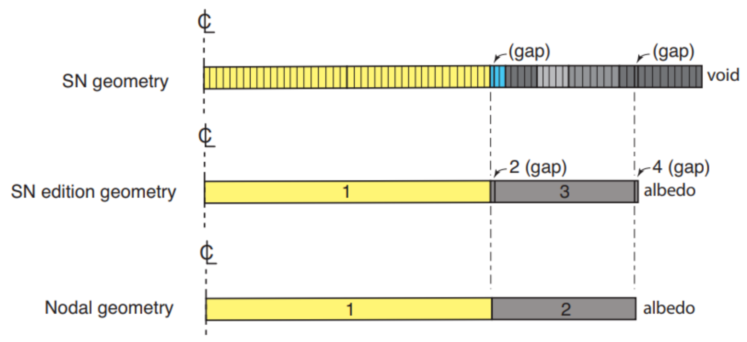


Figure A.2 Geometries used by the Baff-Refl equivalence procedure [9]

a fundamental mode lattice calculation. A fixed geometric buckling is imposed, considering the finite dimensions of the reactor in the transversal directions:

$$B^2 = \left(\frac{\pi}{L_y}\right)^2 + \left(\frac{\pi}{L_z}\right)^2 \approx 10^{-6} cm^{-2} \tag{A.13}$$

Leakages are taken into account by assuming that the group-dependent leakage rate is equal to:

$$L_{i,g} = d_{i,g} B^2 \phi_{i,g} \tag{A.14}$$

Where $d_{i,g}$ are leakage coefficients that can be determined using various models (Golfier-Vergain in SCIENCE). The reference calculation is then homogenized and condensed into a geometry equal to the one to be used for the nodal calculation, except for some gap regions used to compute interface currents and fluxes. An equivalent albedo is added in the homogenized geometry to account for the residual regions in the heterogeneous geometry that are not included in the homogeneous one.

Reflector parameters are then determined in the following way [39]:

- Cross sections are determined by condensation and homogenization from the S_n reference calculation
- Diffusion coefficients are determined by condensation and homogenization the leakage coefficients. The resulting coefficients will depend on the model used.
- Discontinuity factors at the core/reflector using the fluxes obtained from the nodal and reference calculation

This method is a non iterative one. However it is strictly dependent on the feeding assemblies that are provided. In particular, the resulting coefficients are sensitive to the assembly burnup. The Baff-Refl method can be generalized to 2D geometries and to more than 2 energy groups, provided that a reference MOC calculation is available in 2D.

APPENDIX B PRACTICAL EXAMPLE OF LIBRARY USE

B.1 Building the KAIST-like reflector

This section shows a practical use of the library to build a simplified model for the KAIST-like reflector from reference [1]. Only few characteristics of the library are shown to keep the script simple and short for the sake of coinciseness.

B.1.1 Reflector data

The data required to draw the reflector geometry is found in reference [1]. This data is stored in variables that will be used later in the script. Only two refinement variables are defined, however, in principle one refinement variable can be defined for each specific region. Refinement variables are Python dictionaries containing three keywords which are the ALAMOS refinement parameters, where `minSize` and `maxSize` are the minimum and maximum sizes of an edge of the refined mesh. Here, the `refinement2` variable will be used to refine the regions close to the core, to get a better approximation of the flux and reaction rates distribution.

```
################
##### Data #####
################

cell_pitch = 1.26
assembly_pitch = 17 * cell_pitch
bypass_thickness = 0.5
barrel_in_radius = 102.216
barrel_out_radius = 107.931
downcomer_thickness = 24.3026

refl_radius = barrel_in_radius - bypass_thickness
downcomer_size = downcomer_thickness + barrel_out_radius
core_length = 2 * (barrel_out_radius + downcomer_thickness)
hole_radii = 0.7
hole_pitch = assembly_pitch / 5

# Refinement data

refinement1 = {"fineness": 2, "minSize": 0.1, "maxSize": 0.5}
refinement2 = {"fineness": 2, "minSize": 0.05, "maxSize": 0.1}
```

B.1.2 Reflector structure

The reflector's structure is represented by the following items:

- A downcomer, represented with a rectangular shape
- Some circular regions, containing both the baffle and the barrel
- A core shaped region, that replicates the shape of the KAIST core [35]

The first two regions are described by `Rectangle` and `CircularRegions` objects. The core shaped region is instead described by a `CartesianCoreShapedRegion` object. This object must be given a string map, called here `structure_core`, which indicates the core's structure.

```
################################
##### REFLECTOR STRUCTURE ######
################################

### DOWNCOMER ###

downcomer = Rectangle(name=None,
  x1=-downcomer_size,
  y1=-downcomer_size,
  x2=downcomer_size,
  y2=downcomer_size,
  materials="water_MAT",
  temperature="water_TEM",
  ssh_property="Downcomer",
  refiner_options=refinement1)


### CIRCULAR REGIONS ###

circular_regions = CircularRegions(
  name="reflector_circular_regions",
  radius_data={
    barrel_in_radius: {
      "materials": "water_MAT",
      "temperature": "water_TEM",
      "ssh_property": "Byps",
      "refiner_options": refinement1
    },
    barrel_out_radius: {
      "materials": "baffle_MAT",
      "temperature": "baffle_TEM",
      "ssh_property": "Barr",
      "refiner_options": refinement1
    },
    refl_radius: {
      "materials": "baffle_MAT",
      "temperature": "baffle_TEM",
      "ssh_property": "Refl",
      "refiner_options": refinement1
    }
  }
)


### STRUCTURE CORE ###

structure_core = """
  . . A A A A . . \n
  . A A A A A A . \n
  A A A A A A A A \n
  A A A A A A A A \n
  A A A A A A A A \n
  A A A A A A A A \n
  . A A A A A A . \n
  . . A A A A . .\n
  """

expansion_parameters = {
    0: {"materials": "core", "temperature": "", "ssh_property": "", "
                                          refiner_options": ""},
    1: {"materials": "baffle", "temperature": "", "ssh_property": "
                                          expanded_region", "refiner_options":
                                          refinement2}
  }

expanded_regions = CartesianCoreShapedRegions(
  name="expanded_region",
  core_structure=structure_core,
  assembly_size=assembly_pitch,
  expansion_parameters=expansion_parameters
)
```

B.1.3 Engineering components

The only engineering components in the KAIST-like reflector are water channels, whose shape can be described by a `WaterChannel` class. When refining complex geometries like the one found in the KAIST-like baffle where several water holes are present, it can happen that automatic ALAMOS refiner produces very thin and small void regions between the baffle and the water channel [see Annnex—]. This bug is probably caused by numeric error propagation. To accommodate the refining procedure and avoid this bug, a solution can be to insert the water channel in a square box. This feature is already implemented in the `WaterChannel` class and is used in the following code snippet.

```
#####################################
###### ENGINEERING COMPONENTS #######
#####################################

### WATER CHANNELS ###

hole_box = 1 # Side of the square perimeter

# Square perimeter

square_perimeter = Rectangle(name= "square_perim",
    x1= -hole_box,
    y1= -hole_box,
    x2= hole_box,
    y2= hole_box,
    materials= "baffle_MAT",
    temperature= "baffle_TEM",
    ssh_property= "Refl",
    refiner_options= refinement1
)

# radius/refinement dictionaries

channel_water_radius_refinement_dict= {
  hole_radii: "" # water channels not refined
}

# water channel
```

```
w_channel = WaterChannel(name="water_channel",
    radius_refinement_dict=channel_water_radius_refinement_dict,
    outer_perimeter_item=square_perimeter
)
```

B.1.4 Item positioning

For each defined item, one must specify their position to place them in the reflector properly. In this example, the `w_channel` object must be positioned in several locations. By default, `WaterChannel` objects are centered in the origin, and hence, one must translate them to place them in their correct position. To do so, the library provides a built-in function called `translate_in_several_positions_cartesian`, which takes as input a Python list containing the coordinates of all the water channels and returns a list of `ItemPositioningInstruction` objects that will provide the instructions for `WaterChannel` positioning. Other built-in functions are also available, allowing, for example, to provide input data in polar coordinates for translations or to perform translations followed by rotations (useful for `Key` positioning).

```
center_list = [[87.822, 2.142], [87.822, 6.426], [87.822, 10.71],
          [87.822, 14.994], [87.822, 19.278], [87.822, 23.562],
          [87.822, 27.846], [87.822, 32.13], [87.822, 36.414],
          [87.822, 40.698], [87.822, 44.982], [92.106, 2.142],
          [92.106, 6.426], [92.106, 10.71], [92.106, 14.994],
          [92.106, 19.278], [92.106, 23.562], [92.106, 27.846],
          [92.106, 32.13], [92.106, 36.414], [96.39, 2.142],
          [96.39, 6.426], [96.39, 10.71], [96.39, 14.994],
          [96.39, 19.278], [96.39, 23.562], [66.402, 44.982],
          [66.402, 49.266], [66.402, 53.55], [66.402, 57.834],
          [66.402, 62.118], [70.686, 44.982], [70.686, 49.266],
          [70.686, 53.55], [70.686, 57.834], [70.686, 62.118],
          [70.686, 66.402], [74.97, 44.982], [74.97, 49.266],
          [74.97, 53.55], [74.97, 57.834], [74.97, 62.118],
          [79.254, 44.982], [79.254, 49.266], [79.254, 53.55],
          [79.254, 57.834], [83.538, 44.982], [83.538, 49.266]
]

water_channel_positioning = translate_in_several_positions_cartesian(
                                                   center_list)
```

B.1.5 Assembling the Reflector object

The reflector object is finally instantiated. The `reflector_structure` variable is a list containing the odered sequence of items defined in B.1.2, while the `items` attribute is a dictionary where the keys are the engineering components defined in B.1.3 (here only water channels) and the corresponding values is the corresponding list of `ItemPositioningInstruction`. The `symmetry` variable is a keyword that specifies the symmetry of the reflector. Accordingly, here, an eighth of core will be represented by ALAMOS. Output geometry type can be specified using a `OutputOptions` object. Here, the output geometry will be a pin-by-pin one.

```
#####################
##### REFLECTOR #####
#####################

reflector_structure = [downcomer, circular_regions, expanded_regions]

items_dict = {w_channel: water_channel_positioning}
symmetry = "eighth" # reflector symmetry
out = OutputOptions(output_type="pin_by_pin", number_pins_per_side=17) #
                                                  output geometry options



### REFLECTOR ###

reflector = Reflector(
                       name="KAIST",
                       reflector_structure=reflector_structure,
                       items=items_dict,
                       symmetry=symmetry,
                       output_options=out,
                      )
```

B.1.6 Building the geometry

The geometry is built calling the `Reflector.build()` method. With the informations provided when instantiating the `Reflector` class, the library will automatically generate three distinct layers. The first is a refined layer to be used for flux calculations and the second is the output homogenized geometry for the full core and reflector system. the third layer is created by default and represents a non refined geometry to be used for TRIPOLI4® calculations.

```
################################
```

```
#### BUILDING THE GEOMETRY #####
################################

layer, hom_layer, tr4_layer = reflector.build()
```

Figure B.1 shows the flux layer geometry generated by the script. The geometry shows the capability of the `expanded_region` parameter to perform a spatially dependent refinement. Water channels are not refined since no refinement was specified in B.1.3. Figure B.2 shows instead the other geometries that are automatically generated from the procedure. The output geometry in Figure B.2a exhibits, as requested, a pin-by-pin format covering the whole reflector and core geometry. Figure B.2a shows a potential geometry for TRIPOLI4® instead, resulting from merging all the neighboring regions having the same *material* field.

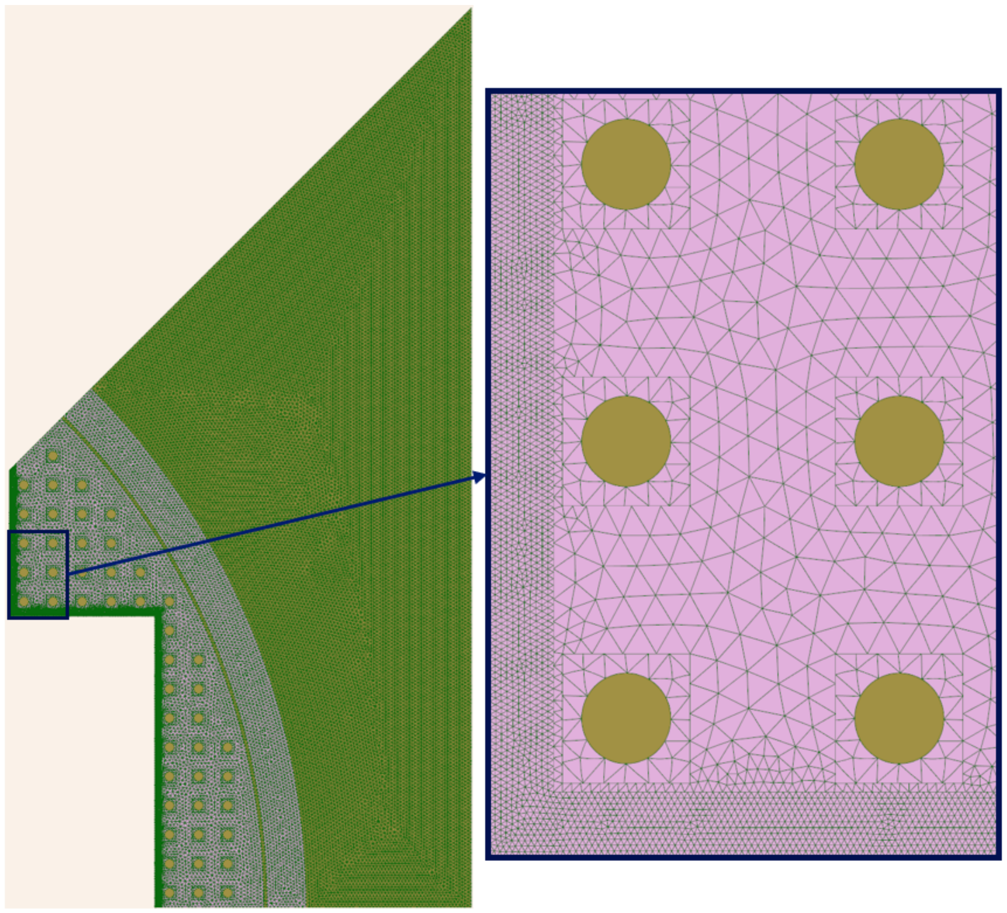
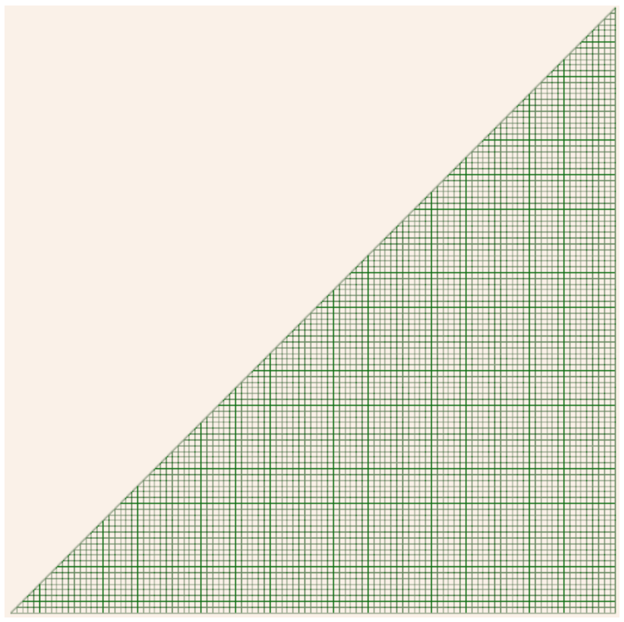


Figure B.1 Flux geometry resulting from the `Reflector.build()` procedure


(a)

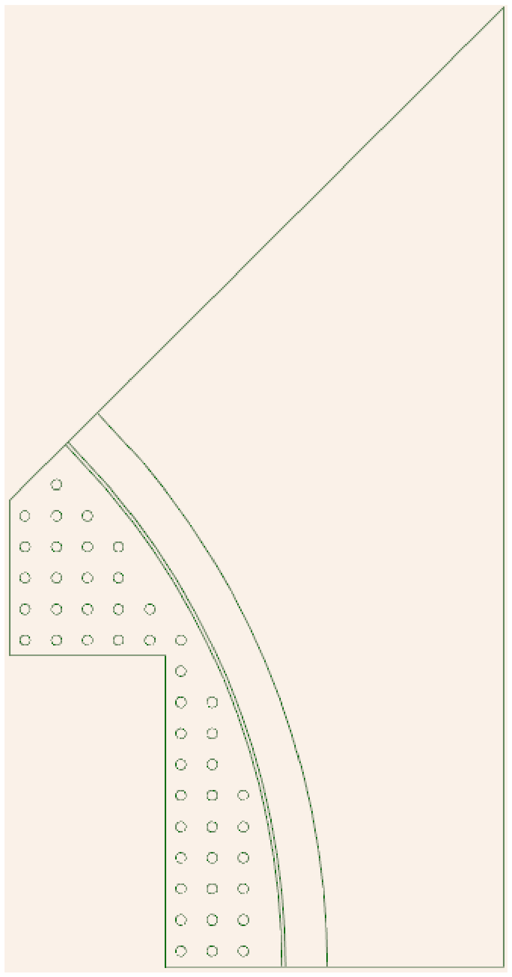

(b)

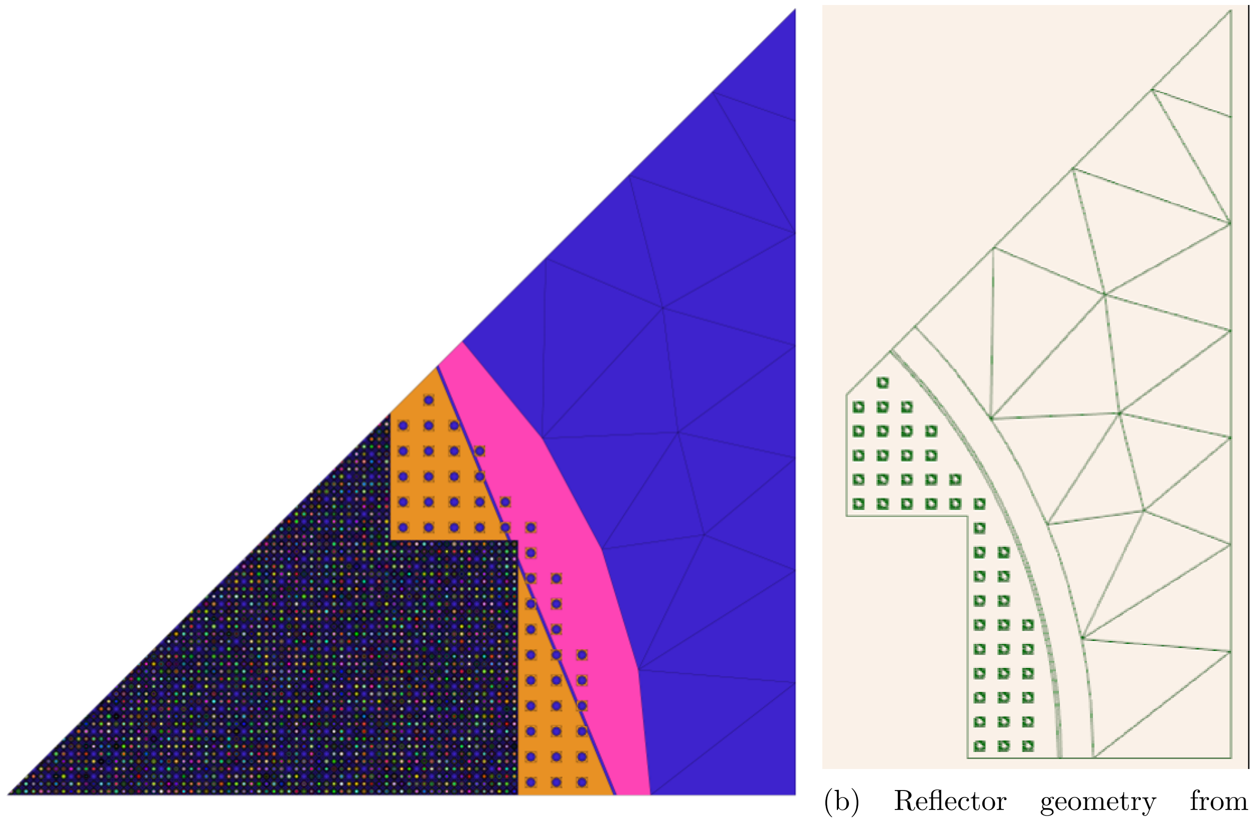
Figure B.2 Geometry of the (a) output and (b) TRIPOLI4® layers resulting from the `Reflector.build()` procedure

APPENDIX C REPORTED BUGS

The present section lists the most interesting bugs and limitations that were found during the present work. It is important to keep track of these bugs and report them to ensure continuous improvements of the codes.

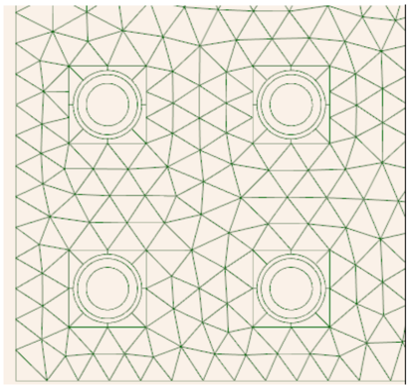
C.1 Native and ALAMOS geometry compatibility

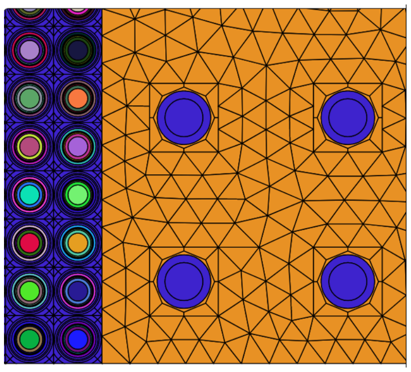
When testing reflector capabilities for the KAIST-like with the NA-RL configuration, a problem related to the way APOLLO3® interprets arcs of circles was found. In fact, if an ALAMOS geometry is appended to a native one (as in the NA-RL case, where a reflector from ALAMOS is appended to a native core), arcs of circles are interpreted as straight lines. Some examples are provided in Figures C.3 and C.2


(a) Geometry from APOLLO3®

(b) Reflector geometry from ALAMOS

Figure C.1 Testing native and ALAMOS geometry compatibility


(a) Geometry from ALAMOS


(b) Reflector geometry from APOLLO3®

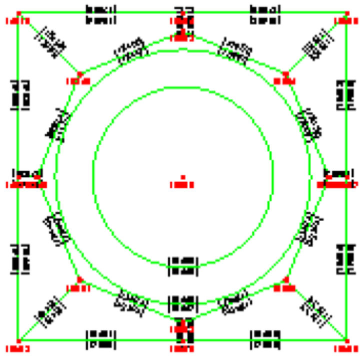

(c) Close-up view of APOLLO3® mesh

Figure C.2 Testing native and ALAMOS geometry compatibility

C.2 Small void regions in ALAMOS

When building complex geometries, such as the one in the EPR reflector's baffle, it may happen that some small void regions are created after refining. These regions do not produce errors in ALAMOS and are so small that can hardly be spotted without knowing exactly where they are. These regions can be spotted by running an APOLLO3® calculation and viewing the mesh geometry: regions close to the void will be highlighted as being part of the core boundaries. Often, to solve this issue, it is sufficient to accommodate ALAMOS refinement by "simplifying" the geometry, for example, by inserting the pin circles in a squared box, as discussed in Paragraph B.1.3.

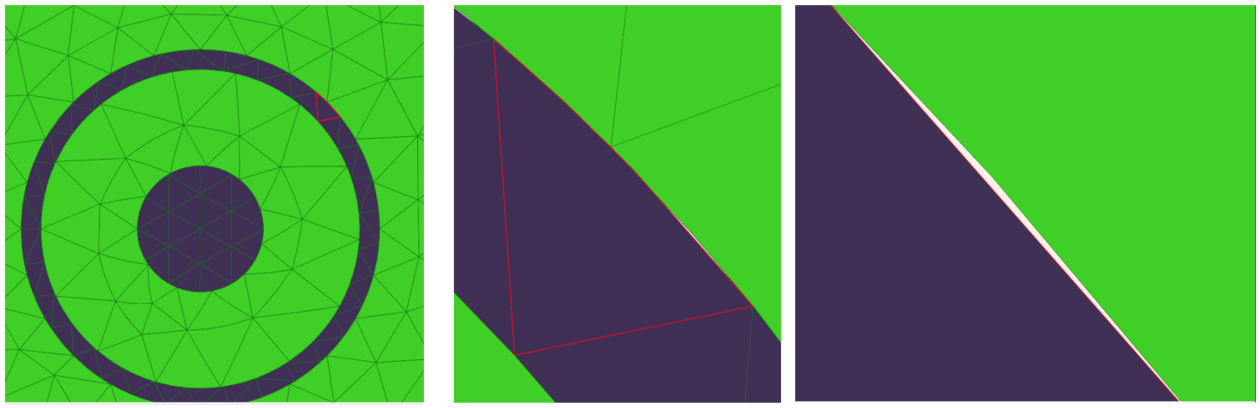


Figure C.3 Void regions in ALAMOS

C.3 Circles in ALAMOS

The previous section already shows one limitation that can be associated to the use of circles in ALAMOS. Another recurrent problem is related to the fact that sometimes ALAMOS struggles building construction lines when trying to draw an object close to circles. An example is given here. Considering the geometry given in Figure C.4a, if one wants to insert an item (here a thermal shield) in the second crown, ALAMOS retrieves an "Unable to build construction lines" errors. The solution, shown in Figure C.4b is to replace the circles with polygons.

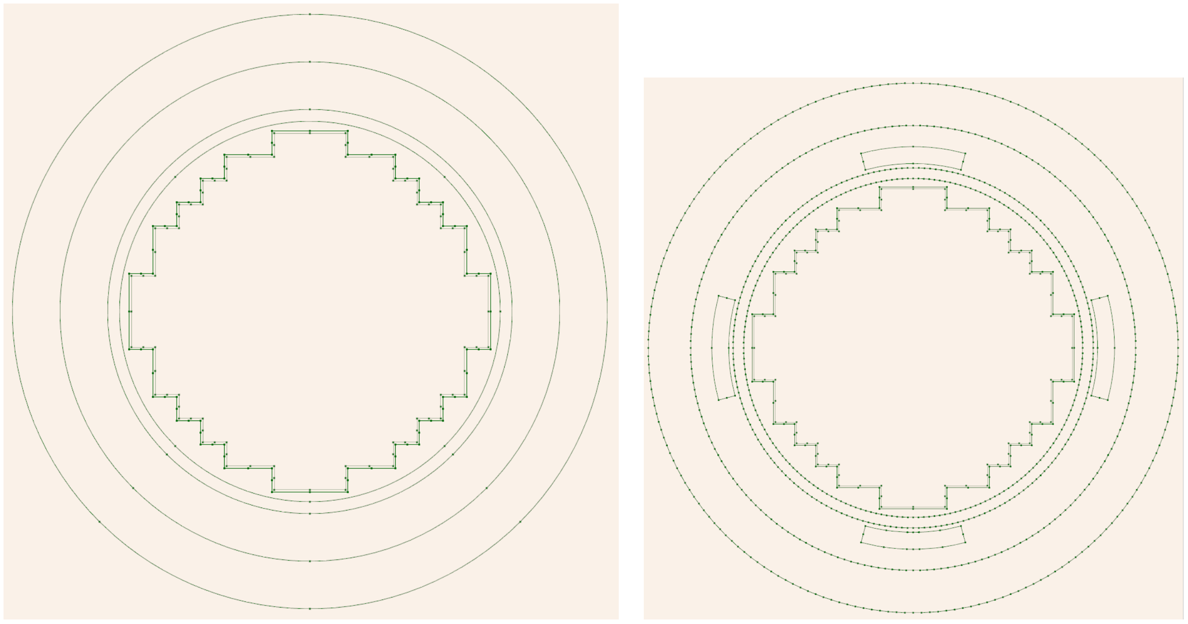
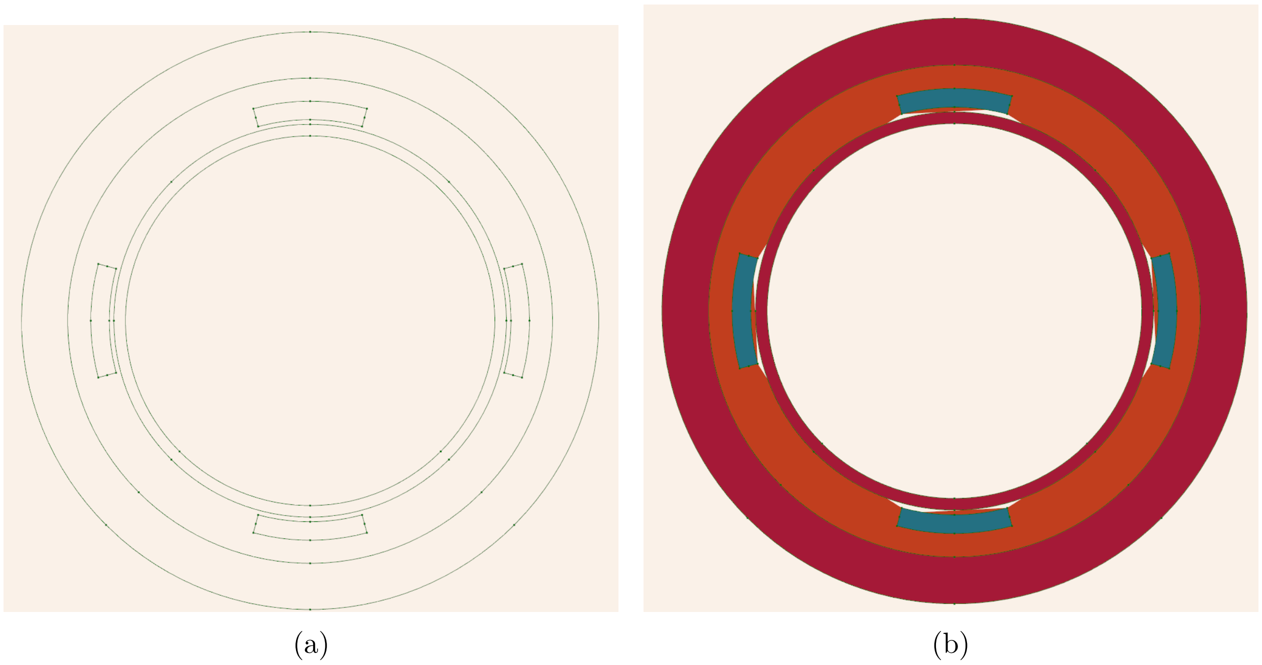

(a) (b)

Figure C.4 Using polygons can help ALAMOS to build construction lines

Interestingly, by removing the core shaped region and keeping only the circular regions, thermal shields are correctly appended. However, by displaying the fields in the geometry some parts of the geometry are left blank.



(a) (b)

Figure C.5 Void regions appearing when appending thermal shields