

INSTITUTO DE ENGENHARIA NUCLEAR

Marco Antonio Chaves Lima

Simulação de um SMR usando OpenMC

Rio de Janeiro
2022

MARCO ANTONIO CHAVES LIMA

SIMULAÇÃO DE UM SMR USANDO OPENMC

Dissertação apresentada ao Programa de Pós graduação em Ciência e Tecnologia Nucleares do Instituto de Engenharia Nuclear da Comissão Nacional de Energia Nuclear como parte dos requisitos necessários para a obtenção do Grau de Mestre em Ciência e Tecnologia Nucleares.

Orientadores: Prof. Dr. Daniel Artur Pinheiro Palma
Prof. Dr. Adino Américo Heimlich Almeida

Rio de Janeiro
2022

SIMULAÇÃO DE UM SMR USANDO OPENMC


Marco Antonio Chaves Lima

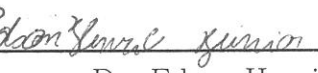
DISSERTAÇÃO SUBMETIDA AO PROGRAMA DE PÓS GRADUAÇÃO EM CIÊNCIA E TECNOLOGIA NUCLEARES DO INSTITUTO DE ENGENHARIA NUCLEAR DA COMISSÃO NACIONAL DE ENERGIA NUCLEAR COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA E TECNOLOGIA NUCLEARES.

Aprovada por:


Prof. Daniel Artur Pinheiro Palma, D. Sc


Prof. Adino Américo Heimlich Almeida, D. Sc


Prof. Cláudio Márcio do Nascimento Abreu
Pereira, D. Sc


Dr. Edson Henrice Junior, D. Sc

RIO DE JANEIRO, RJ - BRASIL
MAIO DE 2022

LIMA Lima, Marco Antonio Chaves

Simulação de um SMR usando OpenMC / Marco Antonio Chaves
Lima. -- Rio de Janeiro: CNEN/IEN, 2022.

xv, 107 f. : il.; 31 cm

Orientadores: Daniel Artur Pinheiro Palma

Adino Américo Heimlich Almeida

Dissertação (Mestrado) – Instituto de Engenharia Nuclear, PPGIEN,
2022.

1.Simulação. 2. SMR. 3. OpenMC. 4.Python

Onde há vida, há esperança.

Henri Charrière (1906-1973)

AGRADECIMENTOS

Quero agradecer a Deus em primeiro lugar, meu companheiro de todas as horas, que a todos os momentos segura nas minhas mãos e nunca me abandona;

À minha mãe, pelo amor, compreensão, apoio incondicional e por sempre interceder por mim em suas orações;

Aos meus irmãos Andréia, Déborah e Marcelo, pelo apoio e pela preocupação em sempre manter meu astral alto;

Aos meus amigos Wanderson Magalhães e André Martins, que me ajudaram de todas as maneiras possíveis nesta jornada;

Ao meu grande amigo Umberto Cassará Siciliano, ex-aluno de Mestrado no IEN e quem me indicou a Instituição;

À minha grande amiga Mônica Luiz Vicente, pelo apoio e ombro amigo;

Ao orientador, Prof. Dr. Daniel Artur Pinheiro Palma, pela receptividade, bom humor, pelo incentivo, confiança (sem nem me conhecer pessoalmente), disponibilidade e apoio;

Ao co-orientador, Prof. Dr. Adino Américo Heimlich Almeida, pelo conjunto da obra: infinita paciência, atenção, disponibilidade, confiança, apoio e por ter ajudado todas as vezes que o código travou sem nenhuma explicação plausível, não importando se era feriado, sábado ou domingo;

Aos amigos que escolheram a área de Tecnologia e Segurança de Reatores: Igor Jaloto, Rodrigo Senra, Luiz Flávio e Edson Minelli (*In Memoriam*);

À secretária do PPGIEN Leila Carla, pelo disponibilidade, cortesia e apoio;

À Prof. Dra. Inaya Correa Barbosa de Lima e à Prof. Dra. Renata Autoum Simão pelo carinho, amizade, incentivo e toda ajuda possível para meu ingresso na pós graduação;

À Prof. Dra. Andressa Nicolau de Métodos Computacionais do PEN e à Prof. Dra. Fernanda Duarte Vilela Reis de Oliveira de Computação II da POLI, professoras que ministraram disciplinas que foram cruciais para o desenvolvimento deste trabalho,

além de excelentes profissionais;

Ao meu orientador da graduação em Engenharia Nuclear, Aquilino Senra Martinez, pelo incentivo e apoio;

Aos amigos Janaína, Thomaz, André, Ivan, Barreto, Carlos, Felipe, Márcia, Vagne, Elias, Aline, Ruth, Alex, Kécia, Zé Carlos pelo apoio, momentos de descontração, ombro amigo, conversa agradável, enfim... essas pessoas fizeram parte do meu alicerce emocional durante essa pandemia;

À diretora da Residência Estudantil, Sheila Imamura, por sempre ter me tratado como um filho e por ter feito o possível e o impossível para me ajudar a concluir o Mestrado;

Ao técnico da TIC, Elison Ferreira, o Gaúcho, pelo apoio, paciência, solicitude e imenso talento para qualquer assunto envolvendo computadores e programação;

Ao meu amigo Luiz Tavares, estudante do curso de Ciência da Computação/UFRJ, pelo imenso talento para programar qualquer coisa e me ajudar no que fosse preciso;

Ao amigo Anselmo Batista, Arquiteto/UFRJ, por sempre me manter sorrindo;

À Comissão Nacional de Energia Nuclear, CNEN, pela apoio financeiro, que sem ele este projeto não seria possível.

SIMULAÇÃO DE UM SMR USANDO OPENMC

Marco Antonio Chaves Lima

RESUMO

Neste trabalho o núcleo de um reator modelado a partir do protótipo argentino CAREM 25 foi simulado com o código de transporte OpenMC, desenvolvido pelo MIT. Mais recente que outros códigos de transporte já estabelecidos como SERPENT e MCNP, o OpenMC é um código open source que tem o Python como API (Application Programming Interface), uma linguagem orientada a objeto muito usada na computação científica. Manipulando objetos através de classes e seus respectivos métodos, o núcleo do reator foi modelado com o sistema CSG, e, depois, os arquivos de input necessários foram construídos. A simulação do núcleo foi utilizada para avaliar o reator proposto de acordo com a região de criticidade determinada pelo valor estimado de k_{eff} fornecido ao final da simulação: menor que 1, *subcrítico*, igual a 1, *crítico*, e maior que 1, *supercrítico*. O reator proposto possui as seguintes características: três tipos de enriquecimento, refrigerado a água leve, utiliza Gd_2O_3 como veneno queimável e foi simulado em BOL (beginning-of-life). Além de estimar o k_{eff} , o código dispõe de outras ferramentas que servem para caracterizar o sistema reacional, como por exemplo: contagens de taxas de fissão, fluxo, taxa de absorção, análise dos dados nucleares, além de permitir analisar como os parâmetros de simulação influenciam os resultados obtidos para o k_{eff} . Com a finalidade de avaliar a performance do código, foi simulado o benchmark BEAVRS, também desenvolvido pelo MIT. Este benchmark consiste num reator PWR convencional com as seguintes características: três tipos de enriquecimento, boro solúvel no refrigerante, silicato de boro como veneno queimável, dois ciclos de carregamento, além da descrição dos materiais estruturais presente no núcleo. A simulação do benchmark comprovou a capacidade do código em simular o comportamento de um modelo já conhecido, com valor estimado do k_{eff} menor que 1.01, levemente supercrítico, como este tipo de reator é projetado para operar. A simulação do reator proposto resultou num sistema supercrítico, com valor de k_{eff} em torno de 1.06, com três regiões de maior fluxo de nêutrons dispostas simetricamente e localizadas nas regiões de maior enriquecimento e sem presença de gadolínio, mostrando que o reator proposto tem um aproveitamento do combustível nuclear diferente de um PWR convencional, como o benchmark, por exemplo.

Palavras-chave: CAREM 25, fator de multiplicação efetivo, Monte Carlo, simulação, OpenMC, Python.

SMR SIMULATION USING OPENMC

Marco Antonio Chaves Lima

ABSTRACT

In this work the core of a nuclear reactor based on the Argentine prototype CAREM 25 was simulated with OpenMC transport code, developed by MIT. More recent than others well-established transport codes like SERPENT and MCNP, OpenMC is an open-source code with Python as API (Application Programming Interface) an object-oriented language largely used in scientific computing. Manipulating objects through classes and their methods, the reactor core was modeled with the CSG system, and then the required input files were built. The core simulation was used to evaluate the proposed reactor according to the critically region determined by the estimated value of k_{eff} provided at the end of the simulation: less than 1, *subcritical*, equal 1, *critical*, and greater than 1, *supercritical*. The proposed reactor has the following characteristics: three types of enrichment, cooled by light water, uses Gd_2O_3 as burnable poison, and was simulated in BOL (beginning-of-life). In addition to estimating of k_{eff} , the code has other tools that can be used to characterize the reaction system, such as: counts of fission rates, flux, absorption rate, nuclear data analysis, in addition to allowing to analyze how the simulation parameters influence the results obtained for k_{eff} . In order to evaluate the performance of the code, the BEAVRS benchmark, also developed by MIT, was simulated. This benchmark consists of a conventional PWR reactor with the following characteristics: three types of enrichment, boron soluble in the coolant, boron silicate as a burnable poison, two charging cycles, in addition to the description of the structural materials present in the core. The benchmark simulation proved the code's ability to simulate the behavior of an already known model, with an estimated value of k_{eff} less than 1.01, slightly supercritical, as this type of reactor is designed to operate. The simulation of the proposed reactor resulted in a supercritical system, with value of k_{eff} around 1.06, with three regions of greater neutron flux arranged symmetrically and located in regions of greater enrichment and without the presence of gadolinium, showing that the proposed reactor has a different use of nuclear fuel than a conventional PWR, such as the benchmark, for example.

Keywords: CAREM 25, effective multiplication factor, Monte Carlo, simulation, OpenMC, Python.

Lista de Figuras

3.1	Esquema do SMR CAREM 25	8
3.2	Esquema de um PWR Clássico	8
3.3	Planta do Circuito Primário de uma central PWR	10
3.4	Geradores de Vapor do CAREM 25	10
3.5	Esquema de Geração de Energia da Central Nuclear CAREM	14
3.6	CAREM 25 - Sistemas Passivos de Segurança	16
3.7	Esquema de uma vareta combustível	18
3.8	Esqueleto do Elemento Combustível	20
3.9	Vista do bocal superior do elemento combustível	21
3.10	Vista do bocal inferior do elemento combustível	21
3.11	Esquema do elemento combustível do CAREM 25	22
3.12	Esquema do núcleo do CAREM 25	23
4.1	Exemplo de herança de classe no OpenMC	27
4.2	Exemplo de polimorfismo das subclasses no OpenMC	28
4.3	Fluxograma da Simulação com OpenMC	32
4.4	Semi-espaço em CSG	36
4.5	Universo Pino	38
5.1	Esquema do pino combustível	47
5.2	Esquema do pino combustível com veneno queimável	47
5.3	Esquema do pino de barra de controle	49
5.4	Divisão dos componentes por um plano de referência	50
5.5	Esquema de inserção de barras de controle	50
5.6	Elemento central do reator	52
5.7	Elemento combustível com veneno queimável	53
5.8	Corte axial de um elemento combustível com veneno queimável	53
5.9	Configurações de elementos combustíveis com veneno queimável	54
5.10	Configurações de elementos combustíveis sem veneno queimável	54
5.11	Esquema de numeração dos elementos combustíveis no núcleo	55
5.12	Distribuição de enriquecimento no núcleo	56
5.13	Sistemas de Controle - bancos de barras de controle	57
5.14	Núcleo do reator, baffle e vaso de pressão	58
5.15	Núcleo do reator modelado no OpenMC	59
5.16	Detalhe do núcleo do reator proposto mostrando o elemento central	59
6.1	Seção do núcleo do benchmark com regiões de enriquecimento - ciclo 1	61
6.2	Posições das varetas com veneno queimável no núcleo do benchmark	64
6.3	Sistemas de Controle do benchmark	64
6.4	Elemento combustível 1.6% de enriquecimento do benchmark	65

6.5	Elemento combustível 1.6% de enriquecimento do benchmark: cortes transversal e longitudinal	66
6.6	Elemento combustível 1.6% de enriquecimento do benchmark: bocal inferior	66
6.7	Elementos combustíveis com veneno queimável do benchmark	67
6.8	Elemento combustível com 15 pinos de veneno queimável do benchmark . .	68
6.9	Corte longitudinal em um elemento com veneno queimável do benchmark .	68
6.10	Elemento combustível com barras de controle do benchmark	69
6.11	Núcleo do benchmark	72
6.12	Ampliação do núcleo do benchmark	73
8.1	Gráfico de Convergência do k_{eff} para o reator proposto	80
8.2	Gráfico de Covergência do k_{eff} para o benchmark	82
8.3	Comparação entre fontes iniciais	83
8.4	k_{eff} obtido pela Equação 8.4 para o benchmark.	84
8.5	Probabilidade de escape ressonante - bechmark.	85
8.6	Fator de fissão rápida - benchmark.	86
8.7	Utilização Térmica do Combustível - benchmark.	86
8.8	Número de nêutrons de fissão produzidos por absorção no combustível - benchmark.	87
8.9	probabilidade de não fuga rápida - benchmark.	87
8.10	Probabilidade de não fuga térmica - benchmark.	88
8.11	k_{eff} pela fórmula dos 6 fatores - benchmark.	88
8.12	k_{eff} obtido pela Equação 8.4 para o reator proposto.	89
8.13	Probabilidade de escape ressonante - reator proposto.	90
8.14	Fator de fissão rápida - reator proposto.	90
8.15	Utilização Térmica do Combustível - reator proposto.	90
8.16	Número de nêutrons de fissão produzidos por absorção no combustível - reator proposto.	91
8.17	probabilidade de não fuga rápida - reator proposto.	91
8.18	Probabilidade de não fuga térmica - reator proposto.	91
8.19	k_{eff} pela fórmula dos 6 fatores - reator proposto.	92
8.20	Filtros de Energia e Malha para as contagens do benchmark.	93
8.21	Contagens de fissão para dois grupos de energia - benchmark.	94
8.22	Fluxo médio do benchmark com malha 100 x 100.	94
8.23	Ciclos de carregamento 1 e 2 do benchmark	95
8.24	Mapa de liberação de energia no núcleo do benchmark.	96
8.25	Filtro de Energia e Malha	97
8.26	Contagens das taxa de fissão para dois grupos de energia - reator proposto.	97
8.27	Fluxo médio e taxa média de fissão do reator proposto.	98

8.28	Seções de choque de captura dos isótopos Gd^{155} e Gd^{157} e de fissão dos isótopos U^{235} e U^{238}	99
8.29	Mapa de liberação de energia no núcleo do reator proposto.	100

Lista de Tabelas

3.1	Dados principais do projeto original do CAREM 25	12
3.2	Tabela comparativa entre as centrais CAREM e PWR	15
4.1	Conceito de abstração	28
5.1	Dimensões da vareta combustível	46
5.2	Dimensões das barras de controle	48
5.3	Dimensões do elemento combustível	51
5.4	Dimensões principais do barrel e vaso de pressão	58
6.1	Carregamento do núcleo do benchmark	61
6.2	Dados principais do EC do benchmark	63
6.3	Controle da reação em cadeia do benchmark	69
6.4	Dimensões dos componentes estruturais do núcleo do benchmark	71
7.1	Simulação do benchmark com 1000 partículas/ciclo com fonte inicial em formato de caixa	75
7.2	Simulação do benchmark com 10000 partículas/ciclo com fonte inicial em formato de caixa	75
7.3	Simulação do benchmark com 100000 partículas/ciclo com fonte inicial em formato de caixa	76
7.4	Simulação do reator proposto com 1000 partículas/ciclo com fonte inicial em formato de caixa	77
7.5	Simulação do reator proposto com 10000 partículas/ciclo com fonte inicial em formato de caixa	77
7.6	Simulação do reator proposto com 100000 partículas/ciclo com fonte inicial em formato de caixa	77
7.7	Simulação do reator proposto com 1000 partículas/ciclo com fonte inicial pontual	78
7.8	Simulação do reator proposto com 10000 partículas/ciclo com fonte inicial pontual	78
7.9	Simulação do reator proposto com 100000 partículas/ciclo com fonte inicial pontual	79
8.1	Valores de k_{eff} vs número de partículas/ciclo - reator proposto.	81
8.2	Valores de k_{eff} vs número de partículas/ciclo - benchmark.	83
8.3	Comparação entre simulação e contagens derivadas para o k_{eff} do benchmark.	88
8.4	Comparação entre simulação e contagens derivadas para o k_{eff} do reator proposto.	92

Lista de Abreviaturas e Siglas

- SMR: Small Modular Reactor
- PWR: Pressurized Water Reactor
- CNEA: Comisión Nacional de Energía Atómica
- CONUAR: Combustibles Nucleares Argentinos
- API: Application Programming Interface
- CSG: Combinatory Solid Geometry
- EC: Elemento combustível
- POO: Programação orientada a objeto
- BEAVRS: Benchmark for Evaluation and Validation of Reactor Simulations
- BOL: beginning-of-life
- RPV: Reactor Pressure Vessel
- LOCA: Loss of Coolant Accident
- RES: Rapid Extension System

Sumário

1	Introdução	1
2	Objetivo e Justificativa	4
2.1	Roteiro da Dissertação	5
3	Projeto CAREM	6
3.1	Circuito Primário	7
3.2	Segurança	13
3.3	Sistemas Passivos de Segurança	15
3.4	Composição do Reator Protótipo CAREM 25	17
3.4.1	Vareta Combustível	17
3.4.2	Barras de Controle	19
3.4.3	Elemento Combustível (EC) - Componentes Estruturais	19
3.4.4	Elemento Combustível (EC) - Composição	21
3.4.5	Núcleo	23
4	OpenMC	24
4.1	Programação Orientada a Objeto - POO	24
4.2	OpenMC: Código de Transporte	30
4.2.1	Arquivo de entrada: <code>materials.xml</code>	34
4.2.2	Arquivo de entrada: <code>geometry.xml</code>	35
4.2.3	Arquivo de entrada: <code>settings.xml</code>	39
4.2.4	Arquivo de entrada: <code>tallies.xml</code>	42
4.2.5	Arquivos de saída: Outputs	43
5	Modelagem do Reator Proposto	45
5.1	Pinos	45
5.2	Elemento Combustível	49
5.3	Núcleo	55
6	Benchmark BEAVRS	60
6.1	Elemento Combustível	62
6.2	Núcleo	70
7	Simulação	74
7.1	Benchmark	74
7.2	Reator Proposto	76

8 Resultados	80
8.1 Convergência do fator de multiplicação efetivo	80
8.2 Fórmula dos 6 fatores para k_{eff}	84
8.2.1 Benchmark	84
8.2.2 Reator Proposto	89
8.3 Contagens de Fissão e Fluxo	92
8.3.1 Benchmark	92
8.3.2 Reator Proposto	96
9 Conclusão	101
ANEXO I	103
Referências Bibliográficas	104

1 Introdução

A demanda por energia elétrica e o crescente interesse pela utilização de fontes renováveis para geração de energia vem reacendendo o papel da energia nuclear no Brasil e no mundo. A diversificação da malha energética brasileira implica numa expansão da oferta de energia econômica e sustentável visando a evolução da demanda, sob uma perspectiva de longo prazo (EMPRESA DE PESQUISA ENERGÉTICA, 2020). Apesar de fazer parte do sistema elétrico nacional, sob forma de Angra 1 e 2, e até 2026, Angra 3, o estigma associado aos perigos da energia nuclear dificulta sua maior aceitação pela população brasileira, (MACHADO and Hansen, 2018). O alto custo de instalação dos reatores, geração de resíduos e o risco de acidentes são também questionamentos de grupos contrários ao uso da energia nuclear (DA VEIGA, 2018).

Os recentes acontecimentos mostram que até mesmo países que aboliram o uso da energia nuclear por pressão de grupos ambientais, já consideram o religamento de suas centrais nucleares. A dependência de fontes fósseis para geração de energia, principalmente para aquecimento das casas durante o período do inverno comprovou o equívoco de desligar suas centrais nucleares (SAIDI and Omri, 2020). Além da vantagem energética da energia nuclear sobre o carvão mineral ou qualquer outra fonte de origem fóssil, a inclusão na categoria de fontes energéticas limpas, atendendo ao critério de não emitir de gases poluentes na atmosfera, recolocou as centrais nucleares no debate energético dos países do hemisfério norte (PRĀVĀLIE and Bandoc, 2018).

No Brasil, o debate energético gira em torno da utilização de termoeletricas na base do sistema elétrico brasileiro e pela crescente preocupação ambiental dos impactos gerados pela utilização de fontes energéticas dessa natureza. Recentes preocupações com os níveis dos reservatórios das hidrelétricas devido à sazonalidade das chuvas, e dos múltiplos usos desses reservatórios também colocaram a matriz hidrelétrica no cerne da questão energética. Tornou-se uma necessidade definir um planejamento energético multifacetado onde as exigências de mitigação de impactos ambientais, a educação da população quanto ao uso racional da energia elétrica e maior participação de fontes renováveis em substitui-

ção às não renováveis formem um tripé no qual o sistema elétrico brasileiro está assentado (DE OLIVEIRA et al., 2021).

A matriz energética brasileira é formada por fontes renováveis correspondentes a 44% do setor (2016) (ALBUQUERQUE et al., 2019). No entanto, o ano de 2021 expôs um ponto crítico da geração hidrelétrica: a falta de chuvas. E as projeções para cenários futuros devem incluir fontes alternativas no sistema elétrico brasileiro para diversificar a oferta de energia elétrica regionalmente, sem correr o risco de ficar restrito às hidrelétricas. Isso recoloca o programa nuclear no planejamento energético do Brasil, pois já ocupa um papel importante da geração de energia no Estado do Rio de Janeiro, importante região metropolitana do país.

O domínio do ciclo do combustível, a disponibilidade de matéria prima, como o urânio, e a expertise acumulada da gerência de Angra 1 e Angra 2 podem servir para alimentar uma futura expansão da fatia nuclear na geração de energia elétrica. O Brasil, apesar da tecnologia utilizada ser baseada em reatores de potência do tipo **PWR** (*Pressurized Water Reactor*), a introdução de reatores inovadores, mais precisamente, de pequenos a médios e/ou reatores modulares, mais conhecidos pela sigla **SMR** (*Small Modular Reactors*), pode aproveitar a cadeia produtiva que dá suportes às centrais nucleares já existentes (EMPRESA DE PESQUISA ENERGÉTICA, 2020).

Reatores de terceira (III+) e quarta (IV) geração são mais conhecidos por avançados ou inovadores. Reatores deste tipo que possuem sistemas de segurança passivos e maior aprimoramento das tecnologias empregadas, implicando em melhor automação e controle, são uma proposta muito mais arrojada de segurança e economia comparada à dos reatores convencionais. Dentro desta categoria, os reatores modulares ocupam um lugar de destaque, pois são muito menores, exigem menor tempo de construção, podem ser fabricados em partes pequenas (ou módulos, daí o nome modulares) e transportados mais facilmente, sem mencionar que podem ser instalados em regiões distantes, ou insulares, podendo servir como mini usina para prover energia elétrica para lugares com demanda sem a necessidade de estender a malha energética já existente para essas localidades (ROWINSKI et al., 2015).

Neste trabalho será feito um estudo de criticidade de um reator baseado em um SMR de fabricação argentina, o CAREM 25. O reator escolhido para servir como modelo representa uma inovação na pesquisa de reatores nucleares na América Latina e pertence a categoria de reatores avançados. Trata-se de um PWR com circuitos primário e secundário integrados, sistemas passivos, sistemas de segurança redundantes e refrigerado à água leve.

Para o estudo proposto será utilizado o código de transporte OpenMC, lançado em 2012. Um código de simulação de Monte Carlo relativamente novo e menos conhecido do que outros códigos já estabelecidos na comunidade científica de pesquisa de reatores nucleares.

2 Objetivo e Justificativa

Os objetivos deste trabalho são:

- Modelar e simular com o OpenMC o núcleo de um reator em **BOL (Beginning-of-Life)** com geometria e composição baseadas no SMR de fabricação argentina CAREM 25;
- Obter o fator de multiplicação efetivo da simulação do reator proposto e realizar as contagens de taxa de fissão e fluxo no núcleo;
- Da mesma forma que para o reator, simular um modelo já estabelecido, o benchmark **BEAVRS**, que consiste num PWR convencional, cuja análise dos resultados da simulação permitirá avaliar o funcionamento do código verificando se corrobora ou não comportamentos conhecidos da simulação de reatores de potência.

O ponto principal deste trabalho é a simulação de Monte Carlo com um código de transporte recente, o OpenMC, que guarda semelhanças com outros códigos mais difundidos na pesquisa de reatores nucleares, como MCNP, MCNPX, e Serpent. Além de ser um código aberto (*open source*), sua interface de programação é o Python, uma linguagem orientada a objeto que também é aberta e muito difundida na computação científica.

A escolha do SMR CAREM 25 é o reconhecimento do avanço da área de tecnologia de reatores na América Latina. De fabricação argentina e desenvolvido pela Comissão de Energia Atômica da Argentina (CNEA), o CAREM 25 apresenta-se como uma proposta de modelo comercial de SMR inédita nos países latino americanos.

Outro ponto importante é a pesquisa/desenvolvimento de reatores avançados ser de extrema importância em países de grande extensão territorial. A instalação de SMRs em regiões afastadas da rede de distribuição pode proporcionar uma maior oferta de energia elétrica de forma segura e econômica, sem mencionar a possibilidade de instalação em plantas industriais que demandam muita energia para manter seu funcionamento.

2.1 Roteiro da Dissertação

Este trabalho está dividido em capítulos que discorrem sobre os seguintes assuntos:

- **Capítulo 3:** revisão bibliográfica do CAREM 25 que servirá de modelo para o reator proposto, destacando a tecnologia empregada, abordando seus princípios de funcionamento, análise de segurança intrínseca do reator, sistemas passivos de segurança e o detalhamento do elemento combustível e seus constituintes;
- **Capítulo 4:** o código de transporte OpenMC, abordando o tema de programação orientada a objeto, interface do Python, construção dos arquivos de input, e output da simulação;
- **Capítulo 5:** configuração do núcleo do reator proposto baseado no núcleo do CAREM 25, com a distribuição dos enriquecimentos, localização dos sistemas de controle, modelagem com o OpenMC dos pinos, das barras de controle, dos tubos de instrumentação, dos elementos combustíveis e do núcleo;
- **Capítulo 6:** modelagem do benchmark **BEARVS** com o OpenMC;
- **Capítulo 7:** Simulação do reator e do benchmark;
- **Capítulo 8:** resultados e análise das simulações: do reator proposto; do benchmark **BEAVRS**; outputs e fator de multiplicação efetivo; pós processamento da simulação mostrando as potencialidades do código;
- **Capítulo 9:** conclusões.

IMPORTANTE: Ao longo de todo trabalho o tema da programação em Python será abordado, mostrando como o conceito de objeto, classe e seus métodos serão implementados na simulação, tanto na construção dos inputs quanto no pós processamento; quando necessário será mostrado a classe e o método utilizado para cada aplicação.

3 Projeto CAREM

CAREM é um projeto da Comissão Nacional de Energia Atômica da Argentina (CNEA), que consiste em projetar, construir e operar uma pequena planta de energia nuclear que possa servir como base para um modelo a ser licenciado. O reator protótipo CAREM 25, juntamente com todos os componentes e sistemas da planta nuclear (vaso de pressão; o edifício central do reator; os geradores de vapor; os elementos combustíveis; os sistemas de segurança e proteção; edifício das turbinas etc) representa um grande avanço para a indústria nuclear argentina (CNEA, 2017). A parceria público-privada das indústrias responsáveis pela fabricação dos componentes (destaque para **CONUAR S.A. - COMBUSTIBLES NUCLEARES ARGENTINOS** e **INVAP S.E**) e órgãos governamentais de pesquisa associados, consolida a trajetória do país no uso da energia nuclear não apenas na utilização de reatores do tipo CANDU, mas na construção e operação da primeira central nuclear de potência com tecnologia exclusivamente argentina. Este êxito põe o país no seleto grupo de países desenvolvedores de tecnologia nuclear.

O reator protótipo CAREM 25 tem o objetivo de validar uma pequena central nuclear para depois, construir e operar plantas iguais, estabelecendo a Central CAREM como proposta energética comercial. Na Argentina já foram desenvolvidos, construídos e vendidos reatores de pesquisa, mas o CAREM 25 é um caso distinto devido às quantidades de energia gerada e combustível utilizado. O nome faz referência à quantidade a potência elétrica que o reator foi projetado para entregar, 25 MW. No entanto, melhorias foram feitas e a potência elétrica foi elevada para 32 MW. Esta quantidade de energia é suficiente para abastecer uma população de 120 mil habitantes na Argentina com os padrões de consumo registrados pelo próprio Ministério de Energia do Governo Argentino (MARCEL et al., 2017, CNEA, 2017).

3.1 Circuito Primário

A característica principal do CAREM 25 é a ênfase na segurança desde o início do projeto: CAREM 25 é considerado intrinsecamente seguro . Os sistemas de segurança de reatores de 2ª Geração foram instalados na etapa posterior ao projeto do reator; a central CAREM é o oposto. Todo o projeto já contém mecanismos passivos de segurança, além dos componentes estruturais já projetados com coeficientes de segurança com redundância de forma a prevenir falhas durante a operação do reator (MAZZI, 2005, MARCEL et al., 2017).

O CAREM 25 é um PWR integrado com tamanho reduzido. Os reatores PWR (Pressurized Water Reactor), constituem quase três quartos dos reatores no mundo inteiro. É um tipo de reator desenvolvido a partir dos anos 1950, portanto de 2ª Geração, que se estabeleceu comercialmente pela facilidade de operação, circuitos primário, secundário e terciário separados, alta disponibilidade do moderador/refrigerante e sistemas ativos de segurança (LAMARSH and BARATTA, 2001).

A Figura 3.1 mostra o esquema do CAREM 25 com os componentes, as regiões de vapor e água leve, e direção do fluxo do refrigerante. A Figura 3.2 mostra o esquema do circuito primário de um PWR clássico de 2 pernas e um loop (pressurizador). Com estas duas figuras é possível visualizar o que diferencia o SMR CAREM 25 e um PWR convencional.

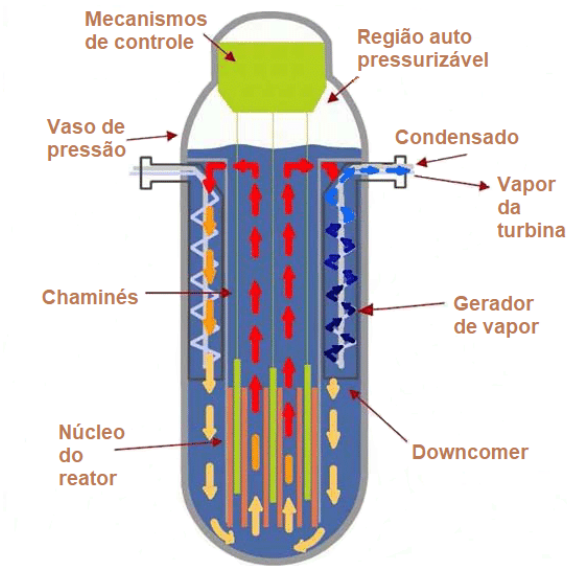


Figura 3.1: Esquema do SMR CAREM 25 - adaptado de MARCEL et al. (2017).

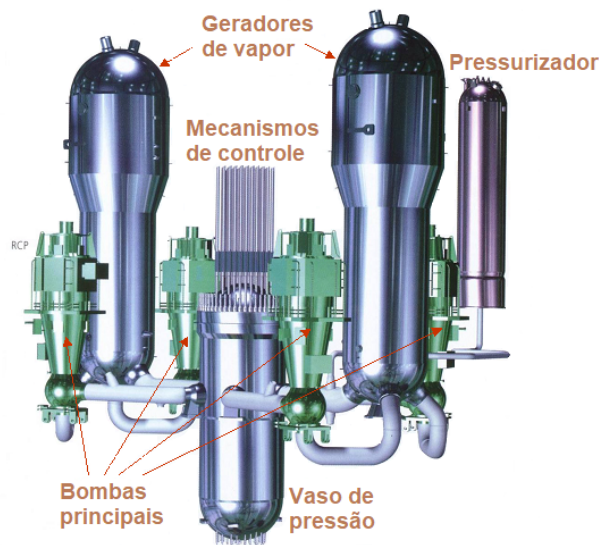


Figura 3.2: Esquema do Primário de um PWR. Ilustração adaptada CNEA.

Ao centro da Figura 3.2, há o vaso pressão do reator, dentro do qual ocorre a reação em cadeia que deve ser controlada ou extinguida mediante a inserção de barras de controle (*Mecanismos de Controle*), localizadas acima do vaso de pressão. Ao redor do *Vaso de pressão do reator* e um acima dele estão dois grandes *Geradores de Vapor*. Nestes componentes o vapor é gerado através do aquecimento da água do secundário sem que haja mistura com a água que circula no primário, para depois, ser direcionado

às turbinas. Na cor verde, quatro *Bombas principais* levam a água do reator até o gerador de vapor, e depois, devolvem-na para o reator, concluindo a circulação. São essas bombas que mais necessitam de energia elétrica. Isso significa que numa eventual pane a circulação de água dentro do reator será interrompida e outro dispositivo para resfriar o núcleo deverá ser acionado. Por último, o *Pressurizador*, que é um componente que tem a função de manter a pressão da água do circuito primário no nível desejado, em torno de 15.5 MPa.

A Figura 3.1 mostra todos os componentes da Figura 3.2 dentro do vaso de pressão do CAREM 25, que possui todos os sistemas integrados ao vaso de pressão, razão pela qual é chamado de **PWR Integrado**. Conseqüentemente, o nível de segurança do CAREM 25 aumenta com todos os componentes de um PWR localizados dentro do vaso de pressão do reator, eliminando a possibilidade de LOCA (*Loss of Coolant Accident*), a necessidade das bombas de refrigeração e do pressurizador, simplificando até mesmo sua operação. Segundo MARCEL et al. (2017), ao incluir os circuitos primário e secundário no vaso de pressão do CAREM 25 diminui-se a quantidade de componentes externos sensíveis, e conseqüentemente, o potencial risco de contato com o ambiente que esses componentes oferecem.

A Figura 3.3 apresenta o esquema da Planta do Circuito Primário de um PWR mostrando o núcleo, onde se localizam os elementos combustíveis, a bomba de circulação (na forma de catavento), o pressurizador e o gerador de vapor. As flechas em azul escuro indicam o sentido do refrigerante. Comparando com a Figura 3.1, observa-se que o CAREM 25 incorpora dentro de seu vaso todos os componentes do circuito primário de um PWR: bomba, pressurizador, geradores de vapor e barras de controle.

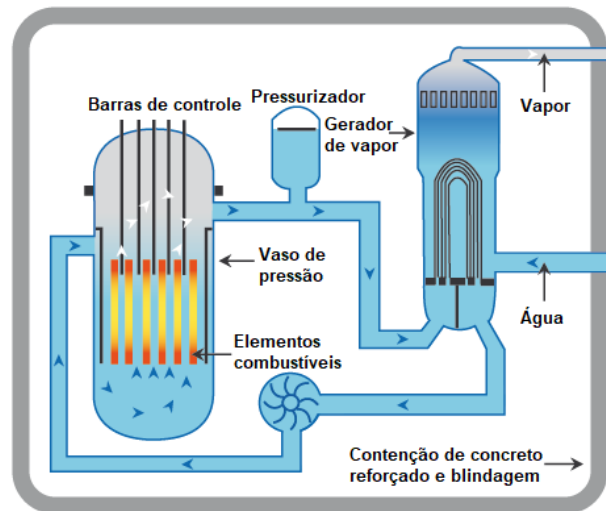


Figura 3.3: Planta do Circuito Primário de um PWR - adaptado de WORLDNUCLEARASSOCIATION (2016).

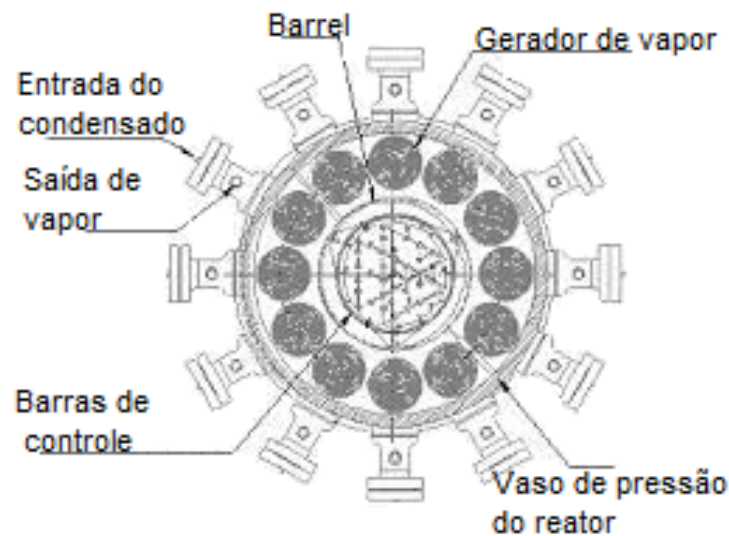


Figura 3.4: Doze Geradores de Vapor do CAREM 25 - adaptado de MAGAN et al. (2014).

Os geradores de vapor estão todos dispostos na parte de cima da periferia do vaso de pressão, como mostra a Figura 3.4. A bomba é incorporada não como um equipamento, mas substituída por sistema passivo: o fenômeno de convecção natural. Basicamente, consiste no movimento do fluido pela variação de densidade em função da temperatura. À medida que a água passa pelo núcleo do reator, ela ganha entalpia devido ao calor

recebido das reações de fissão, dirigindo-se aos geradores de vapor e, posteriormente, às turbinas. Quando a água aquece sua densidade diminui. Depois do vapor passar pela turbina e se condensar, retorna ao núcleo como líquido mais frio, cuja densidade é mais alta que a água que se dirigiu aos geradores. Então, pela disposição do núcleo na parte de baixo do reator (responsável por aquecer a água), e dos geradores na parte de cima (produtores do vapor que depois será resfriado), cria-se uma corrente de circulação sem necessidade de bomba. A água se move pelo princípio físico da diferença de densidade.

O pressurizador do PWR foi incorporado como uma cúpula hemisférica de vapor no topo do vaso de pressão, indicado na Figura 3.1 como ***Região auto pressurizável***, e serve manter a pressão próxima à saturação, equivalente ao *set-point* do pressurizador do PWR. Aliado ao grande volume de água, a auto pressurização mantém o reator sob estabilidade operacional, equilibrando a pressão mesmo durante transientes ou picos de potência (MAGAN et al., 2014).

A Tabela 3.1 sumariza os principais dados do projeto original.

Parâmetro	Dado
Tipo de Reator	PWR Integrado
Potência Elétrica (MWe)	25
Potência Térmica (MWth)	100
Vida útil (anos)	60
Refrigerante / Moderador	água leve
Circulação do primário	Circulação Natural
Pressão (MPa)	12.25
Mecanismo principal de controle de reatividade	Bancos de barras de controle
Altura do vaso de pressão (m)	11
Diâmetro do vaso de Pressão (m)	3.2
Temperatura de saída do refrigerante (°C)	326
Temperatura de entrada do refrigerante (°C)	284
Ciclo Termodinâmico	Ciclo Rankine
Sistemas Passivos de Segurança	Possui
Sistemas Ativos de Segurança	Possui
Tipo de combustível / Arranjo	pastilhas de UO ₂ / hexagonal
Comprimento ativo (m)	1.4
Número de ECs	61
Enriquecimento (%)	1.8 e 3.1
Queima (GWd / tonelada)	24
Ciclo do recarga (meses)	14
Sistema de Segurança Emergencial	Sistema Passivo
Sistema de Remoção de Calor Residual	Sistema Passivo
Diferencial	Remoção de calor por circulação natural

Tabela 3.1: Dados principais do projeto original do CAREM 25, adaptado de DELMASTRO (2021).

3.2 Segurança

A eliminação de componentes externos ao vaso do reator resultante da integração dos mesmos, diminuindo o número de tubulações comparados a circuito primário de um PWR, Figura 3.1. Não há perna quente ou fria no CAREM 25. Isso é uma vantagem porque num PWR uma eventual falha dessas pernas poderiam ocorrer acidentes onde haveria perda de refrigerante, mais conhecidos como LOCA. A eliminação dessas ligações do vaso com outros componentes do circuito elimina este problema.

Os mecanismos de controle (aranhas das barras) são acionados hidraulicamente. Na Figura 3.1, em verde e no topo (onde lê-se: *Mecanismos de controle*), as barras são mantidas na posição vertical. Em reatores PWR são acionados por bobinas eletromagnéticas, mas quando inseridas em um tipo de reator onde as temperaturas podem atingir até 350°C, as propriedades eletromagnéticas são diferentes, por isso não é possível este tipo de controle. Os mecanismos de acionamento das barras são relativamente simples e baseados no fluxo de água que passa por um pequeno orifício controlado por um pistão que, na interrupção deste fluxo, fazem as barras caírem por gravidade. Este simples mecanismo de funcionamento não dependente de energia elétrica e garante a segurança de operação do reator numa eventual falha humana ou num transiente qualquer (MAGAN et al., 2014).

A Figura 3.5 mostra o esquema da central nuclear CAREM 25 disponibilizado pela própria CNEA. Idêntico ao PWR, utiliza água de rio ou mar para condensar o vapor que movimenta as turbinas (item 5).

CENTRAL NUCLEAR CAREM: ESQUEMA DE GERAÇÃO DE ENERGIA ELÉTRICA

- 1 No núcleo encontram-se os elementos combustíveis. Ali a reação de fissão controlada aquece a água do primário no interior do vaso de pressão.
- 2 A água quente do primário eleva a temperatura da água que atravessa o gerador de vapor (circuito secundário). Este vapor movimenta a turbina.
- 3 A turbina faz girar o gerador elétrico, que transforma energia cinética (movimento) em energia elétrica.
- 4 A eletricidade é transportada através de linhas de alta tensão até chegar ao consumidor final.
- 5 A água natural (rio, lago ou mar) refrigera a água do secundário. Este resfriamento condensa o vapor, que regressa como água

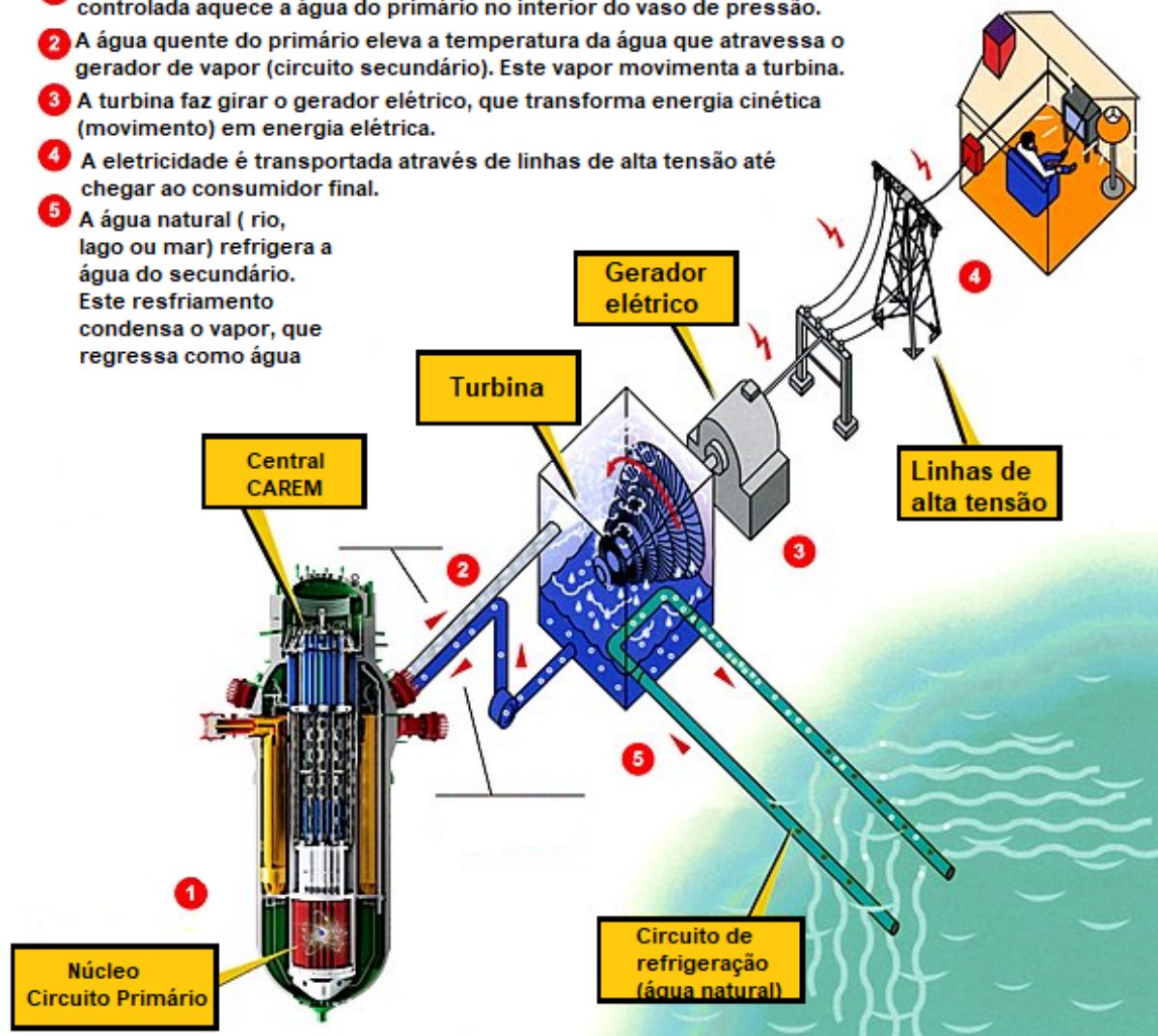


Figura 3.5: Central Nuclear CAREM. Ilustração adaptada CNEA.

A Tabela 3.2 sumariza as diferenças entre duas centrais: PWR convencional e CAREM. Quanto ao tipo de circuito primário (a água que está em contato com o núcleo), na central CAREM é integrado, pois a água nunca sai do reator para geração de vapor, exceto para fins de purificação. Numa central PWR convencional, a água sai do núcleo e depois regressa, por isso o circuito é externo. A pressurização na central PWR é feita por um pressurizador externo, enquanto a central CAREM é auto pressurizável. A circulação do refrigerante na central PWR é realizada por uma bomba movida por energia elétrica

enquanto na central CAREM é feita por circulação natural. O acionamento das barras de controle é hidráulico na central CAREM e por bobinas eletromagnéticas na central PWR. E, por último, na central CAREM todos os sistemas de segurança são passivos, e na central PWR, são ativos.

CENTRAL	PWR CONVENCIONAL	CAREM
CIRCUITO PRIMÁRIO	EXTERNO - LOOP	INTEGRADO
PRESSURIZADOR	EXISTE	NÃO EXISTE - (AUTO PRESSURIZÁVEL)
CIRCULAÇÃO DO REFRIGERANTE	FORÇADA - USO DE BOMBA	NATURAL - SEM USO DE BOMBA
BARRAS DE CONTROLE - TIPO DE ACIONAMENTO	ELETROMAGNÉTICO - EXTERNO	HIDRÁULICO - INTERNO
SISTEMAS DE SEGURANÇA	ATIVOS	PASSIVOS

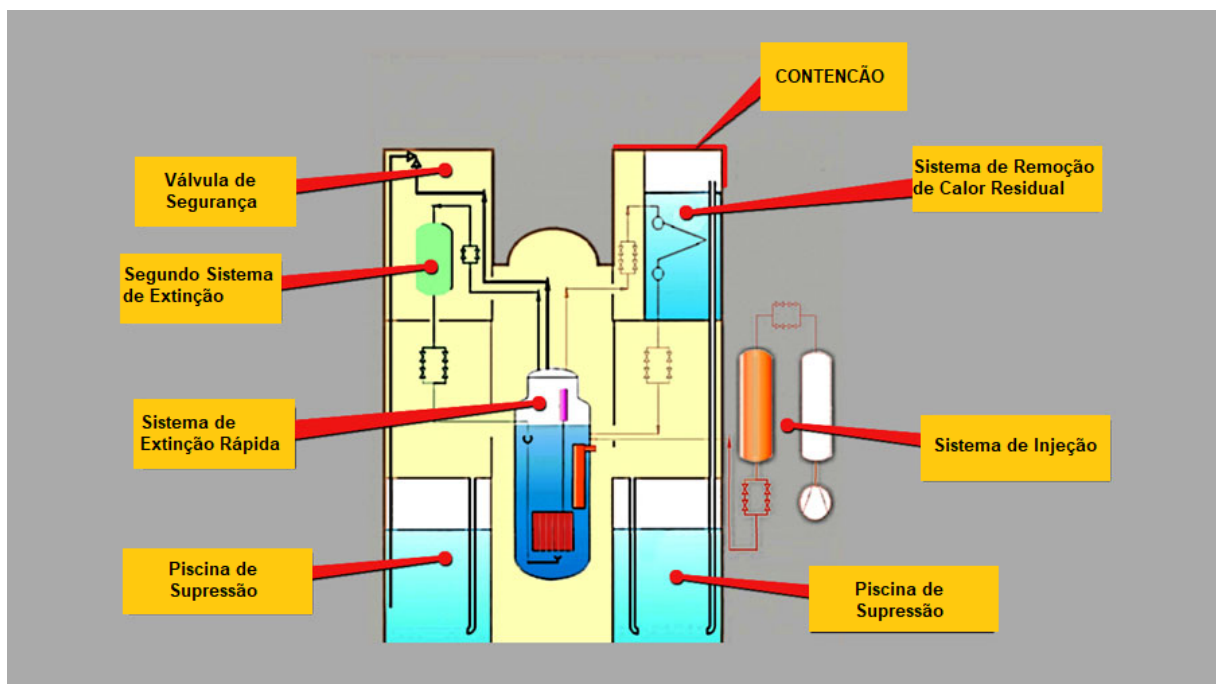
Tabela 3.2: Tabela comparativa entre as centrais CAREM e PWR Convencional.

3.3 Sistemas Passivos de Segurança

A Figura 3.6 apresenta o esquema dos sistemas passivos do CAREM 25. O *Sistema de Extinção Rápida* (considerado também como Primeiro Sistema de Extinção) é representado pelas barras de controle que caem no núcleo por gravidade no caso de falta de energia elétrica. O *Segundo Sistema de Extinção* contém uma solução de ácido bórico cuja função é apagar rapidamente o reator quando a válvula é aberta. Situa-se acima do reator, justamente para cair por gravidade. Uma vez apagado, contudo, devido a natureza das reações que acontecem no núcleo o reator continua gerando calor. Então, torna-se necessário remover esse calor. Um PWR convencional teria uma bomba e um trocador de calor externo, o CAREM 25, no entanto, dispõe do *Sistema de*

Remoção de Calor Residual, situado acima do reator, de modo que o vapor gerado pelo calor liberado nas reações suba até o sistema e se condense, retornando como água em menor temperatura, formando assim um fluxo onde não é necessário nenhum tipo de bombeamento para manter a água do reator dentro dos valores de temperatura que são desejáveis.

No caso de perda do refrigerante o **Sistema de Injeção** é acionado. Este sistema contém águas de média e baixa pressão (são dois tanques), com válvulas que se abrem quando a pressão do lado dos tanques é superior à do reator, fazendo a água fluir para o reator e repor parte o refrigerante perdido. A água que é perdida se vaporiza dentro da **Contenção**, que é seco, aumentando a pressão do edifício. A **Piscina de Supressão** serve para que o vapor, ao tomar contato com a água da piscina, condense e a pressão do edifício de contenção fique normalizada.



CAREM 25: ESQUEMA DOS SISTEMAS DE SEGURANÇA

Figura 3.6: Corte da estrutura de contenção do CAREM 25 com os sistemas de segurança. Do topo, no sentido horário: Contenção; Tanques do Sistema de Remoção de Calor Residual; Tanques do Sistema de Injeção; Piscina de Supressão (são duas legendas pois a ilustração é um corte; a piscina é circular); Barras de Controle (Barras de Desligamento do reator); Sistema de Desligamento Secundário (solução de ácido bórico); Válvulas de Segurança, que ventam na piscina de supressão. Ilustração adaptada CNEA.

Todos os sistemas anteriormente descritos são redundantes e garantem até 36 horas de condição operacional segura, sem haja necessidade de nenhuma intervenção do operador numa eventual falha de um deles ou falta de energia elétrica (MARCEL et al., 2017). Este tempo é suficiente para corrigir qualquer falha ou restituir o fornecimento de energia (ARENAZA, 2018). Como todas as centrais nucleares dispõem de geradores a diesel para acionar um sistema ativo para qualquer emergência (como uma bomba, por exemplo), os sistemas da Figura 3.6 conferem uma segurança muito maior na operação do reator sem suprimento de energia elétrica ou de cenários mais graves, como eventos sísmicos (MAGAN et al., 2014).

3.4 Composição do Reator Protótipo CAREM 25

Nas subseções anteriores foram destacados os aspectos do projeto original do reator protótipo CAREM 25 do ponto de vista da engenharia de reatores. Esta subseção discorre sobre os componentes do núcleo do reator: varetas, barras de controle, tubo de instrumentação, material estrutural etc, destacando apenas os materiais que os compõem e como estão localizados no núcleo. Como o reator que será simulado tem como base o CAREM 25, as dimensões de seus componentes serão aproveitadas e utilizadas para modelagem do reator proposto no código; no Capítulo 5, na modelagem do reator proposto, estas dimensões serão apresentadas.

3.4.1 Vareta Combustível

O CAREM 25 é um reator a água leve e utiliza pastilhas de urânio enriquecido. As varetas combustíveis empregadas no CAREM 25 são de três tipos:

- Carregadas com pastilhas de UO_2 com baixo enriquecimento, 1.8% em peso de U^{235} ;
- Carregadas com pastilhas de UO_2 com alto enriquecimento, 3.1% em peso de U^{235} ;

- Carregadas com dois tipos de pastilhas: de UO_2 misturado a veneno queimável Gd_2O_3 no meio; nas pontas, 20cm abaixo do topo e 10cm a partir da base, pastilhas de UO_2 com 3.1% enriquecimento. Esta configuração heterogênea tem o objetivo de não aumentar o fator de pico (FP) da vareta com veneno queimável, VILLARINO et al. (2012).

A Figura 3.7 mostra o esquema da vareta combustível dos reatores de Angra 1 e 2. O CAREM 25 utiliza a vareta com esta mesma configuração, diferindo apenas nas dimensões, uma vez que é um SMR. O revestimento da vareta é o mesmo, de Zircaloy e duas pastilhas refratárias de alumina estão localizadas nas extremidades inferior e superior da região com pastilhas de dióxido de urânio.



Figura 3.7: Vareta combustível - Ilustração Eletrobrás.

3.4.2 Barras de Controle

O CAREM 25 utiliza como elementos de controle pellets (pastilhas) da liga Ag-In-Cd, nas proporções em peso de 80% Ag, 15% In e 5% Cd. Este material é bem estabelecido como absorvedor de nêutrons em barras de retores PWR. O material de revestimento das barras de controle é o aço inoxidável AISI 316L, um tipo de aço já estabelecido no setor nuclear como material de revestimento para barras de controle e outros componentes estruturais.

As barras de controle tem configuração igual às varetas de combustível (Figura 3.7): com pastilhas de Ag-In-Cd, e revestimento de AISI 316L, no lugar de pastilhas de UO_2 e revestimento de Zircaloy.

3.4.3 Elemento Combustível (EC) - Componentes Estruturais

Do ponto de vista estrutural, o elemento combustível consiste de um esqueleto formado pelos tubos guia, tubo de instrumentação, 4 grades separadoras e os bocais inferior e superior, Figura 3.8. Acima do bocal superior está a aranha das barras absorvedoras. Os tubos guia e de instrumentação são fixados aos bocais inferior e superior por parafusos. As grades separadoras são soldadas nos tubos guia em posições ao longo do comprimento para dar estabilidade ao conjunto; o fluxo de água impõe uma vibração aos elementos combustíveis muito grande. O esqueleto abriga também as varetas, fixando-as nas grades espaçadores através de juntas elásticas, formada por quatro pontos de apoio e uma mola.

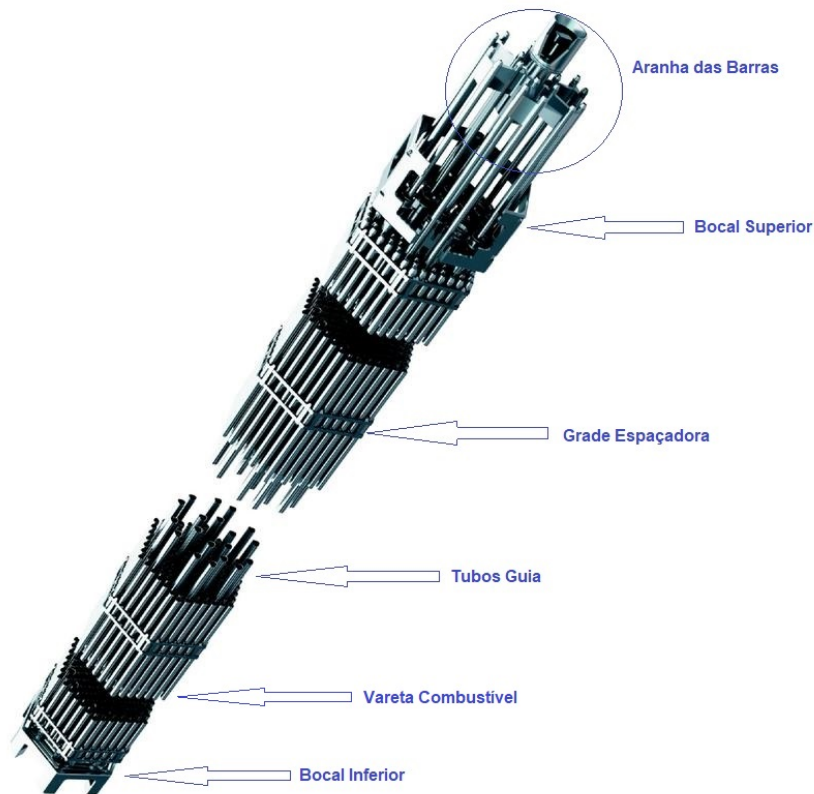


Figura 3.8: Elementos estruturais do EC CAREM 25. Em destaque, a aranha dos bancos de barras, o bocal superior, grade espaçadora, tubos guia circundados por varetas combustíveis e bocal inferior. Adaptado de MAGAN et al. (2014)

A aranha consiste num conjunto de barras absorvedoras que se movem como uma única unidade. Elas servem para inserir as barras por dentro dos tubos guia que caem por empuxo, pois os tubos estão cheios de água. As barras são utilizadas no controle da reatividade do reator durante a operação normal através dos Sistemas de Ajuste e Controle. A inserção é feita em passos fixos (*steps*), isto é, cada passo é uma parte do comprimento das barras que estará presente no núcleo. O número necessário será determinado pelo operador para manter o reator crítico. No caso de emergência, todos os bancos caem simultaneamente desligando o reator (DELMASTRO et al., 2010).

Os bocais inferior e superior, grades espaçadoras, mangas das grades, tubo de instrumentação, são também feitos de AISI 316L. A Figura 3.9, da vista do bocal superior, mostra os tubos guia por onde as barras de controle são inseridas no núcleo pela aranha (vide Figura 3.8). Na Figura 3.10, a vista do bocal inferior de um elemento combustível.

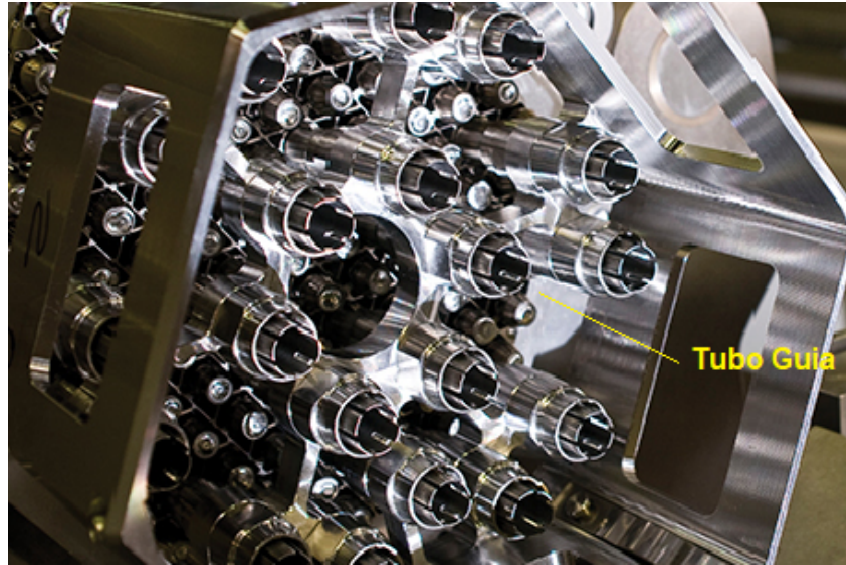


Figura 3.9: Vista do bocal superior de um elemento combustível do CAREM 25. Em destaque, os tubos guia por onde são inseridas as barras de controle. Imagem CONUAR.

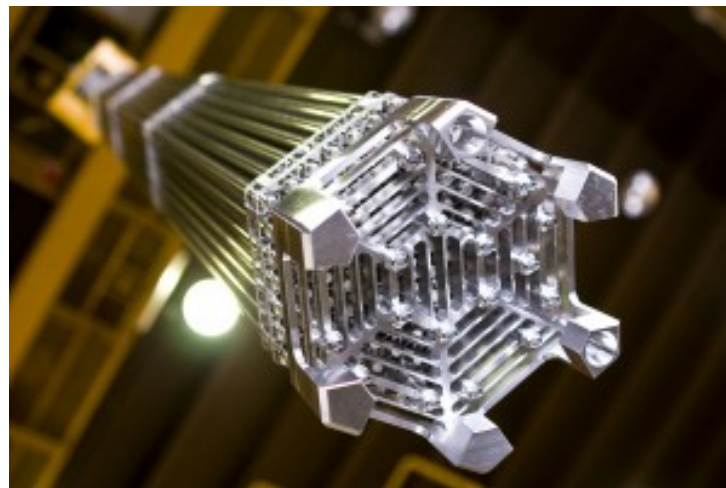


Figura 3.10: Vista do bocal inferior de um elemento combustível do CAREM 25. Imagem CONUAR.

3.4.4 Elemento Combustível (EC) - Composição

A disposição das varetas combustíveis, barras de controle e dos tubos de instrumentação no elemento combustível está mostrada no esquema da Figura 3.11. A forma do EC é hexagonal e cada um dispõe de 108 posições com vareta combustível, 18 com tubos guias e uma para instrumentação, totalizando 127 posições. Os componentes dos ECs são

constituídos de materiais já utilizados em outros reatores a água leve . A reatividade do núcleo é controlada pela presença de veneno queimável na forma de gadolínio (Gd_2O_3) misturado ao combustível de maior enriquecimento em posições específicas no EC, e com elementos absorvedores móveis na forma de barras, pertencentes aos Sistemas de Ajuste e Controle, que são inseridas no núcleo pelos tubos guia.

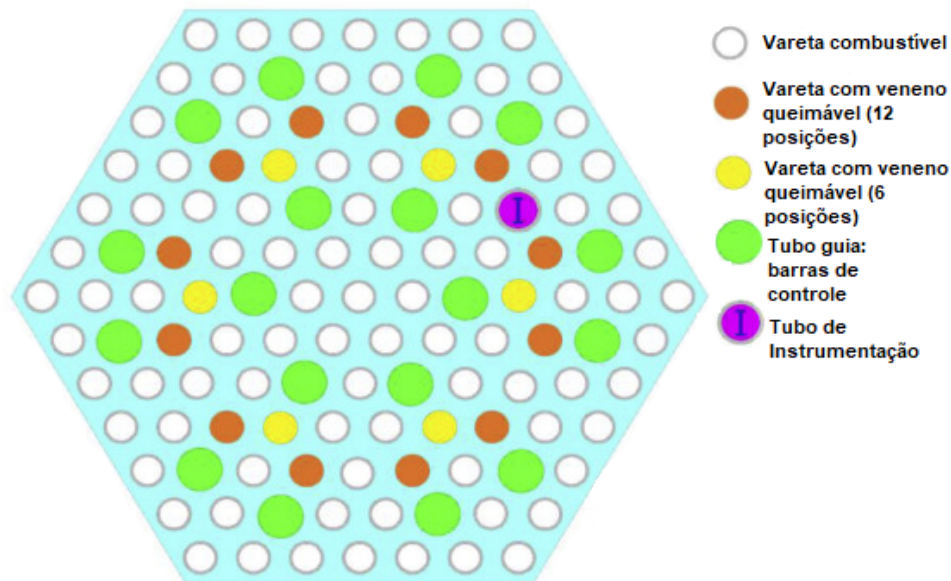


Figura 3.11: Esquema das posições das varetas combustíveis com apenas UO_2 (na cor branca), das varetas com UO_2 e gadolínio (dependendo do EC, ocupando 6 ou 12 posições), das barras de controle (inseridas nos tubos guia) e do tubo de instrumentação num elemento combustível do CAREM 25. Adaptado de TASHAKOR et al. (2017).

De acordo com a Figura 3.11 os ECs podem ter 18 posições possíveis de veneno queimável. No entanto, os ECs podem ter nenhuma, 6 ou 12 varetas com veneno queimável; as varetas com combustível puro ocupam as posições de veneno queimável de forma a completar as 108 posições de combustível.

As 18 posições de tubos guia definem ECs com nenhuma, 12 ou 18 barras de controle que pertencem aos Sistemas de Ajuste (**SA**), Controle (**SC**) e **RES**. Na ausência de um sistemas de controle os tubos guia ficam cheios de água.

Em cada EC há apenas um canal para instrumentação, contudo não necessariamente há instrumentação em todos os ECs.

3.4.5 Núcleo

O núcleo do reator protótipo CAREM 25 possui 61 elementos combustíveis de acordo do esquema da Figura 3.11, com 1.4 m de comprimento ativo, circundado água leve pelo barrel e, mais externamente, pelo vaso de pressão, ambos feitos do mesmo material, o aço AISI 316L. Não há baffle circundando o núcleo do reator, nem blindagem ao redor do barrel. A Figura 3.12 é um corte do CAREM 25 mostrando o barrel, o vaso de pressão, na cor azul, o refletor, água, e os elementos combustíveis que compõem o núcleo. A configuração do núcleo do CAREM 25, que servirá como base para modelar o núcleo do reator proposto, mostrando a localização dos elementos com veneno queimável e com bancos de barras de controle será discutida no Capítulo 5, na modelagem do reator.

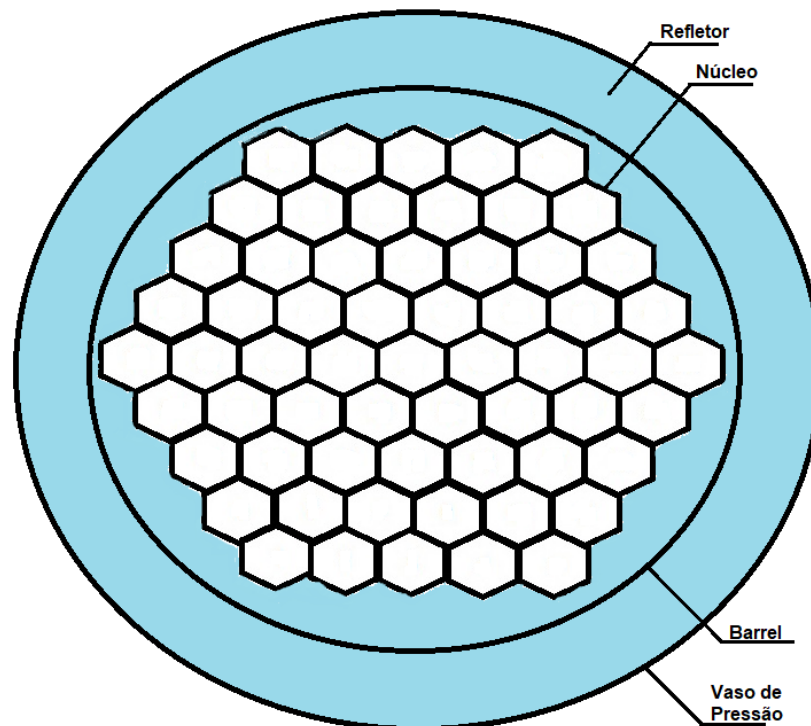


Figura 3.12: Esquema do núcleo do CAREM 25. Marco(2022).

4 OpenMC

Para melhor entender o funcionamento do código de transporte OpenMC este capítulo será dividido em 2 partes:

- Python: linguagem script orientada a objeto muito usada na computação científica;
- OpenMC: o código de transporte (simulador de Monte Carlo).

4.1 Programação Orientada a Objeto - POO

O desenvolvimento de um programa pode ser feito com diferentes linguagens de programação, cada qual com seu paradigma de programação. O paradigma de **Orientação a Objeto** é o mais difundido. Trata-se de um padrão no desenvolvimento de qualquer aplicação moderna, devido a manutenção, segurança e aproveitamento do código.

Dos paradigmas de programação existentes, três exemplos abarcam as linguagens mais conhecidas: Estrutural, Funcional e Orientado a Objeto.

Paradigma Estrutural: Presente nas linguagens **C**, **Pascal** e **FORTRAN**, este paradigma aplica procedimentos aplicados em blocos, e a comunicação entre eles se dá pela passagem de dados (SINTES, 2002). O objetivo deste tipo de programação é desenhar um conjunto de funções que, sequencial e iterativamente, realizam uma tarefa específica. Caso essas funções sejam muito grandes elas serão desmembradas em outras menores. Neste paradigma, as informações são armazenadas em variáveis globais.

Paradigma Funcional: Presente nas linguagens **LISP** e **Haskell**, neste tipo não existem declaração de variáveis, apenas funções. As operações se resumem à composição dessas funções, com amplo uso da recursividade, que é a possibilidade de uma função chamar a si mesma, desde que os parâmetros necessários sejam a ela passados (MUELLER, 2019).

Paradigma da Orientação a Objeto: Presente nas linguagens **Java**, **C++**, **C#** (lê-se: “C sharp”), **Delphi** e **Python**, este é o paradigma mais difundido. Segundo

SINTES (2002), neste tipo de paradigma, os dados e funções relacionados são encapsulados num mesmo elemento, denominado **objeto**. Diferente do Paradigma Estrutural, a comunicação entre objetos se dá pelo envio e recebimento de mensagens. O objetivo da Paradigma Orientado a Objeto é desenhar **classes** que descrevem objetos semelhantes e que atendem a tarefas específicas (SANTOS, 2003).

Apesar do paradigma estrutural dominar o mercado de desenvolvimento de softwares por um longo tempo, com prós e contras, ele foi bastante eficiente e ainda é utilizada em muitas linguagens amplamente difundidas, como C, por exemplo. Embora as linguagens orientadas a objetos existam desde a década de 1960, o crescimento sem paralelo no uso e na aceitação de tecnologias por objeto tem sido reconhecido por todo o setor de software (SINTES, 2002).

Algumas vantagens da programação orientada a objeto:

- Reaproveitamento do código: separa um problema em partes menores, mais eficientes e autônomas;
- Facilidade de Manutenção: qualquer modificação é feita apenas numa parte de interesse, sem que o alterar as outras partes (MEDEIROS, 2004);
- Confiabilidade e Gerenciamento no código: separado o problema em partes, sua atualização se torna mais dinâmica. No caso do Python, que é um *open source*, diversas versões são lançadas na web constantemente por desenvolvedores individuais ou grupos, tornando a ferramenta sempre atualizada, eficiente (com diminuição de *bugs*) e com mais funções disponíveis aos usuários.

Para justificar o uso é preciso perguntar: o que torna uma linguagem de programação em orientada a objeto? Mais especificamente, do que se trata uma orientação a objeto? Quatro conceitos formam os pilares da linguagem orientada a objeto:

ABSTRAÇÃO: é o ponto mais importante da linguagem orientada a objeto. Trata-se de uma representação de um objeto real dentro do sistema (daí o emprego do nome paradigma). E uma vez representado, será preciso descrever seu comportamento dentro do sistema. Três pontos descrevem seu comportamento:

- **Identidade:** quando o objeto é criado, imediatamente uma identidade é criada no sistema. Ela é única para que não haja conflito toda vez que o objeto é acionado;
- **Propriedade:** um objeto real possui características que o definem. Dentro da programação, essas “características” são denominadas **propriedades**;
- **Método:** as ações que o objeto executa são denominadas **métodos**. Pode ser o método “Falar()” do objeto Pessoa, ou “Acender()” do objeto Luz, por exemplo.

ENCAPSULAMENTO: é a forma com que o programa é dividido a ponto de tornar suas partes mais isoladas possível. Ou seja, cada um de seus métodos são executados isoladamente e retornam um resultado ou uma informação que não pode ser executada para cada situação. Sendo assim, o objeto não conhece a forma que seu(s) método(s) foi(foram) implementado(s). É como se fosse a caixa preta de um avião. O Python, por exemplo, implementa o encapsulamento dos métodos de suas classes (SINTES, 2002). Evita-se o acesso às propriedades dos objetos e permite adicionar uma camada de segurança, uma vez que não é possível acionar diretamente uma propriedade do objeto. Usando como exemplo o objeto Luz, quando o método “Acender()” é acionado, não sabemos como a Luz acende, isto é, o usuário não tem como saber o que acontece internamente na Luz, mas sabe que a Luz acende quando o método “Acender()” é acionado.

HERANÇA: o reuso do código é conhecido como **herança**. A reutilização implica numa redução das linhas de código e do tempo de processamento. A herança permite que objetos herdem características de outros objetos. O objeto Ferrari herda de Carro suas características, pois uma Ferrari é um carro. O objeto Pessoa herda características de Mamíferos, pois uma pessoa (um ser humano) é um mamífero, e assim por diante.

POLIMORFISMO: é a ideia do camaleão adaptada à programação. O camaleão muda (ação) a cor da pele de acordo com o ambiente quando percebe que está em perigo, camuflando-se e distraindo um possível predador. O animal se modifica à medida que muda de ambiente (diferentes ações). Na programação, um objeto herda de um “ancestral” suas características e métodos. Mas as ações para um mesmo método podem ser diferentes. Os objetos Bicicleta e Carro são dois meios de transporte que possuem

o método “Acelerar()”. Mas os dois não pode ser acionados do mesmo jeito, devem ser reescritos para cada caso. Para (SINTES, 2002), o polimorfismo altera o funcionamento de um método interno herdado de um objeto pai.

Esses 4 conceitos permitem a representação de objetos do mundo real em termos de uma classe. Isso significa aproximar o sistema que está sendo criado no ambiente virtual com características e ações que o objeto real possui. A estrutura chamada Classe agrupa objetos com características em comum e descreve as ações possíveis e quais informações ela pode armazenar. Para exemplificar o conceito de classe com o Python, a linguagem adotada pelo OpenMC, vamos analisar algumas classes definidas no código.

A classe `openmc.Surface` do OpenMC define a superfície de um objeto e sua condição de fronteira. Ela contém um conjunto de características que definem uma superfície. No entanto, a ela não possui um método de implementação, pois cada tipo de superfície tem atributos específicos. Ela é uma classe abstrata. Nas classes derivadas de `openmc.Surface`, chamadas **subclasses**, os métodos para cada tipo de superfície serão implementados. Na Figura 4.1, as subclasses utilizadas para gerar as superfícies dos elementos do reator proposto constituem um exemplo mais simples de herança e polimorfismo, herdando características da classe pai, `openmc.Surface`. Na Figura 4.2, cada uma das subclasses possui um atributo específico que a difere das outras, ao mesmo tempo que herdam todos os três atributos da classe `openmc.Surface`: a identidade (**surface_id**), um número inteiro atribuído; o tipo de superfície (**boundary_type**: **transmissiva**, **reflexiva**, **vácuo**, **periódica ou branca**); e o nome da superfície (**name**).

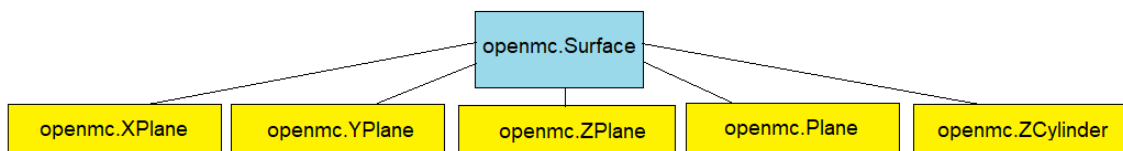


Figura 4.1: Herança de classe no OpenMC. Marco (2022).

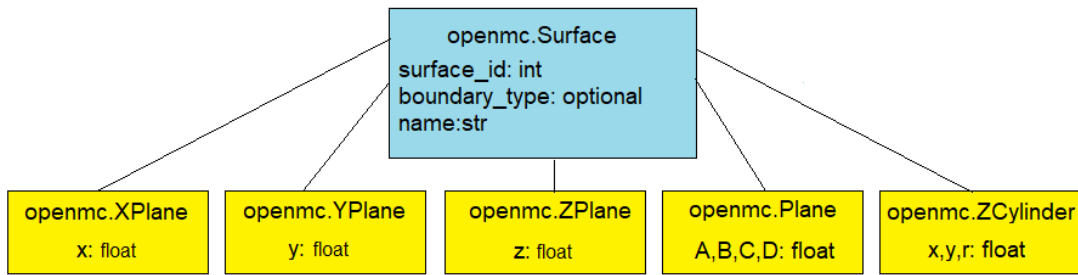


Figura 4.2: Polimorfismo no OpenMC; tipo de dado de entrada dos atributos da classe pai e das classes filhas. Marco (2022).

A Tabela 4.1 descreve duas entidades (objetos) do mundo real, com suas respectivas características.

Objetos	Atributos (características)	Métodos (ações)
Pastilha Combustível	raio, altura, composição, peso	fissão, absorção, nêutrons
Absorvedor	raio, altura, composição, peso	absorção

Tabela 4.1: Abstrações do mundo real.

Na Tabela 4.1 vemos como isso funciona: não se sabe como serão implementadas as contagens de fissões, absorção ou geração de nêutrons (prontos e/ou rápidos), mas sabemos quais contagens fazer para cada objeto e o resultado esperado da mesma. A pastilha combustível é o material físsil. Não é preciso dizer que as contagens de fissão serão feitas na pastilha, pois o objeto que sofre fissão é conhecido e abstrai toda essa informação por trás disso.

O encapsulamento no OpenMC é exemplificado na classe `openmc.Material`, com seus métodos: `add_nuclide` e `add_element`. Ambos podem ser executados isoladamente e retornar ou não um resultado satisfatório, ao mesmo tempo em que é possível atribuir valores que expressam a composição dos materiais de forma independente, sem alterar o funcionamento do restante da classe.

Tomando como exemplo esta mesma classe, o esquema abaixo mostra como um material é declarado no OpenMC. O objeto real é a pastilha combustível; a classe é `Material`, e seus atributos são nome e identidade; os métodos são `add_nuclide` e `add_element`. Importante salientar que o objeto é criado virtualmente quando uma instância da classe é criada. A instância `fuel` é um exemplo de objeto cujo tipo é alguma classe e os métodos são implementados pelo operador “.”

```
fuel = openmc.Material(material_id=1, name='UO2-1.8%')
fuel.add_element('U', 1.0, enrichment=1.8)
fuel.add_nuclide('O16', 2.0)
fuel.set_density('g/cm3', 10.0)
```

O método `add_nuclide` permite especificar apenas o(s) nuclídeo(s) de interesse; no caso, apenas o isótopo 16 do oxigênio. Quando o método `add_element` é utilizado todos os nuclídeos de urânio (U) disponíveis na biblioteca de dados nucleares serão utilizados. Utilizando os dois métodos simultaneamente a proporção atômica entre U e O continua sendo 1:2. E para ser implementado, o material físsil (ou fissionável dependendo do nuclídeo), o método requer um atributo percentual de enriquecimento. O método `set_density` tem como atributos o valor da densidade do material e a sua unidade.

Os conceitos expostos de criação de objetos e a manipulação destes pelos respectivos métodos explica a adoção do Python como API do OpenMC, assim como o uso das linguagens orientadas a objeto no mercado de softwares e aplicativos de celular devido ao aumento da velocidade de processamento de computadores e microcircuitos atuais. A linguagem Java é a preferencial em quase todos os dispositivos eletrônicos portáteis (SANTOS, 2003). A linguagem C# é dominante nos computadores pessoais que possuem sistema operacional Windows, da Microsoft. A substituição das rotinas e sub-rotinas por uma divisão de blocos de dados resulta num código com menos linhas de programação e solidifica o sucesso da programação orientada a objeto (POO). Esta divisão em blocos é mais conhecida como **Modularização** (SANTOS, 2003).

4.2 OpenMC: Código de Transporte

OpenMC é um código de transporte desenvolvido pelo **Massachusetts Institute of Technology (MIT)** e lançado ao público em 2012. O OpenMC simula nêutrons e fótons, problemas de fonte fixa ou do tipo k-eigenvalue, com as seções de choque em multi grupos ou em modo contínuo de energia. A geometria do sistema pode ser por modelada por **CSG (Constructive Solid Geometry)** ou **orientada por malha de polígonos**, permitindo construir modelos em 2D ou 3D de reatores nucleares ou qualquer outro sistema de interesse, como fantasmas para cálculo de dose, simulação de blindagens etc (OPENMC-Contributors, 2020).

O OpenMC simula partículas neutras (fótons e nêutrons) movendo-se aleatoriamente num modelo definido que representa um objeto real (ROMANO and Forget, 2013). É exatamente disso que se trata a simulação de Monte Carlo. Para o caso do objeto ser um reator nuclear, que é o caso deste trabalho, ou qualquer outro sistema físsil, os nêutrons tem um papel multiplicador, pois ao induzirem fissões colidindo com isótopos do urânio ou outros nuclídeos físséis, aumentam sua população dentro do sistema, e consequentemente, o comportamento médio dessa nova população determinará a frequência das reações de fissão e servirão para avaliar a criticidade deste sistema. Esta é a importância da simulação de Monte Carlo no estudo de reatores nucleares.

Por tratar-se de um experimento estatístico as quantidades de interesse são resultados do comportamento médio das partículas simuladas. Isso significa que simulando um grande número de partículas a estimativa do comportamento médio melhora cada vez mais. Estatisticamente falando, é a precisão do experimento que aumenta: sua variância diminui, VUOLO (1996). O Teorema Central do Limite garante que a variância da média amostral de uma quantidade ou grandeza qualquer estimada por Monte Carlo será inversamente proporcional ao número de partículas simuladas, isto é, uma simulação com alta precisão é feita com um número de partículas suficientemente grande para gerar contagens com variâncias menores - em Monte Carlo o número de partículas é também denominado **história de nêutrons**.

Para simulação de reatores nucleares, o tema deste trabalho, estimadores de colisão (**Collision**), absorção (**Absorption**) e caminho percorrido (**Track-Length**) serão usados como estimativas do fator de multiplicação efetivo, além de um estimador combinado de mínima variância baseado na matriz de covariância dos três estimadores, chamado **Combined**. O fator de multiplicação efetivo, k_{eff} , é um parâmetro que descreve quantitativamente a evolução da reação em cadeia, estabelecendo três regimes de controle reacional: menor que 1, *regime subcrítico*, igual a 1, *regime crítico*, e maior que 1, *regime supercrítico* (DUDERSTADT, 1976, LAMARSH and BARATTA, 2001). É um parâmetro de extrema importância na pesquisa de Física de reatores nucleares. No entanto, é importante ressaltar que nem todos os tipos de reatores podem ser simulados com este código. Apesar de simular problemas de fonte fixa, o OpenMC ainda não está habilitado para cálculo de sistemas subcríticos ($k_{\text{eff}} < 1$), como é o caso de reatores ADS, (OPENMC-Contributors, 2020). O fluxograma da Figura 4.3 mostra as etapas da simulação com o código.

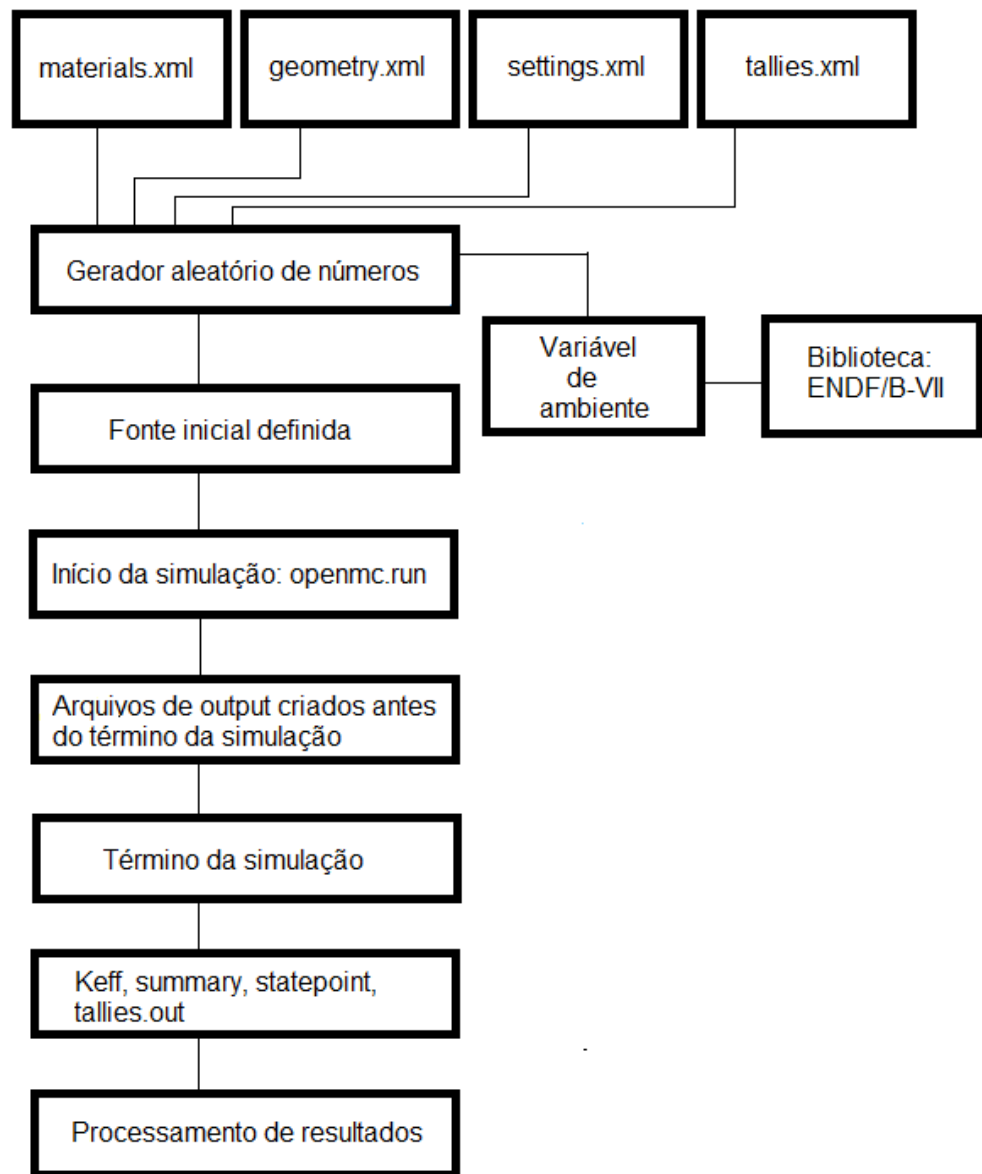


Figura 4.3: Fluxograma da Simulação com OpenMC, Marco (2022).

A simulação com OpenMC utiliza o Python como API (Application Programming Interface). Trata-se da linguagem que o usuário vai utilizar para programar a simulação, daí o nome interface de programação. Como explicitado anteriormente, a vantagem de utilização dessa linguagem está no fato de ser estruturada em classes que permitem manipular objetos que são partes do sistema de interesse, tornando a representação no código mais simples e lógica.

O ambiente no qual o código é escrito é o **Jupyter Notebook**. Este editor de texto em forma de blocos possibilita programar a simulação de forma mais limpa e organizada sem que seja necessário escrever tudo num único bloco para testá-lo depois. Ao contrário, escrevendo em blocos que podem ser testados separadamente, e sem precisar rodar todo o programa, a programação se torna mais eficiente, pois qualquer erro pode ser acessado de forma independente, permitindo a localização de erros à medida que a programação é feita sem a necessidade do programa todo ser rodado. A esses blocos dá-se o nome de **células**, e cada uma delas pode fornecer um output quando executada; é o teste de cada célula do programa.

Depois de instalado, o Jupyter é acessado num navegador de internet, mas funciona localmente, sem a necessidade do usuário estar conectado à rede. A criação de um notebook contendo as células em branco para serem programadas é feita na página inicial do Jupyter. Instalando os pacotes necessários básicos para programar em Python (NumPy, Matplotlib, SciPy, pandas etc) os arquivos de simulação já podem ser escritos.

O funcionamento do código é intuitivo, e para simular um sistema de interesse as etapas básicas são: construir os arquivos básicos de entrada (**inputs**) e alocar os dados das seções de choque numa variável de ambiente (biblioteca de dados ENDF/B-VII). Terminada a simulação, a etapa de pós processamento é particular à análise do sistema em questão, ficando a critério do usuário decidir quais resultados esperados e como tratar os dados gerados da simulação. Na parte final deste trabalho, o pós processamento será detalhado.

Os arquivos de entrada do OpenMC são todos no formato XML (**Extensible Markup Language**), uma linguagem de marcação que serve para definir formatos e padrões de exibição dentro de um documento (semelhante ao **HTML**, formato comumente utilizado na construção de páginas da web); regra a codificação dos documentos na forma em que serão visualizados pelo usuário e pelo código no momento da simulação (SANTOS, 2003, CAMARÃO, 2003). Alguns códigos de Monte Carlo utilizam apenas um arquivo de input, enquanto no OpenMC ele é dividido em várias partes, de forma que cada um corresponda a objetos de uma classe.

Toda simulação requer os seguintes arquivos básicos:

- `materials.xml`: quais materiais estão presentes (na forma de núclídeos e/ou elementos) e suas respectivas densidades;
- `geometry.xml`: a geometria do modelo a ser simulado usando CSG; aponta os materiais para as células;
- `settings.xml`: arquivo que contém os parâmetros de simulação: número de partículas, modo de simulação (fonte fixa, eigenvalue), número de ciclos (batches), tipo de fonte (pontual, caixa etc) e outros parâmetros podem ser ligados ou desligados (atributos do tipo booleano **True** or **False**).

Esses arquivos são guardados na memória do programa ao final de cada etapa quando são exportadas com a extensão xml. Para os três casos acima, podemos relacioná-los com suas respectivas classes:

```
materials.export_to_xml() → classe: openmc.Materials
geometry.export_to_xml() → classe: openmc.Geometry
settings.export_to_xml() → classe: openmc.Settings
```

4.2.1 Arquivo de entrada: `materials.xml`

O arquivo de materiais é constituído de objetos da classe `openmc.Material`, que possui os métodos `add_nuclide`, `add_element`, e `set_density`, como descritos em 3.1. Os núclídeos/elementos que são declarados neste documento devem constar no arquivo que contém as seções de choque dos núclídeos. Este arquivo está guardado na variável de ambiente criada, que é acionada internamente no início da simulação quando é lido o arquivo de materiais. Assim sendo, se o isótopo 18 do oxigênio for declarado e não constar nenhum dado de sua seção de choque (como na biblioteca de dados ENDF/B-VII), o programa apontará um erro. É importante verificar previamente a composição do sistema e as seções de choque disponíveis na biblioteca utilizada; para este trabalho, a biblioteca será a ENDF/B-VII.

O OpenMC utiliza a aproximação do gás ideal para descrever o comportamento dos núcleos alvos quando não estão em repouso, isto é, suas velocidades seguem uma distribuição Maxwelliana. Isso significa que para o espalhamento térmico de nêutrons interagindo com elementos ligados a moléculas, como hidrogênio ou deutério à molécula de água, as seções de choque de espalhamento térmico devem ser usadas. Seções deste tipo (escreve-se: $S(\alpha, \beta, T)$), são arquivos separados dentro da biblioteca de dados utilizada. Como exemplo, quando o material **water** é criado, o método `add_s_alpha_beta('c_H_in_H2O')` deve ser implementado da seguinte forma:

```
water.add_s_alpha_beta('c_H_in_H2O')
```

Uma vez criados todos os objetos na memória, para criar o arquivo `materials.xml` é necessário criar uma instância de `openmc.Materials` que recebe como um atributo a lista dos materiais criados (**IMPORTANTE:** esta não é a classe com que foram criados os “materiais” individualmente). Abaixo, as instâncias dos materiais `fuel` e `water`, que compõem a lista:

```
materials = openmc.Materials([fuel, water])
```

O arquivo `materials.xml` é finalmente criado com o método `export_to_xml()` e implementado como:

```
materials.export_to_xml()
```

4.2.2 Arquivo de entrada: `geometry.xml`

O objeto que aponta um material para uma região no OpenMC é chamado **Cell**. Quando uma região é definida um volume é criado. Então, é necessário primeiro definir quais as superfícies limitam essa região. No OpenMC, os volumes dessas regiões são representados usando o sistema CSG (em português, **Geometria Combinatória**), que consiste em criar **semi-espacos** usando os operadores booleanos: `&` (interseção), |

(união), e \sim (complemento).

Se um volume físico é definido pelas superfícies que o limitam é necessário definir primeiramente uma superfície. Tomemos como exemplo o hemisfério definido na Figura 4.4c, e as equações cartesianas do plano perpendicular a z (Equação 4.1) e da esfera (Equação 4.2):

$$z - z_0 = 0. \quad (4.1)$$

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - R^2 = 0. \quad (4.2)$$

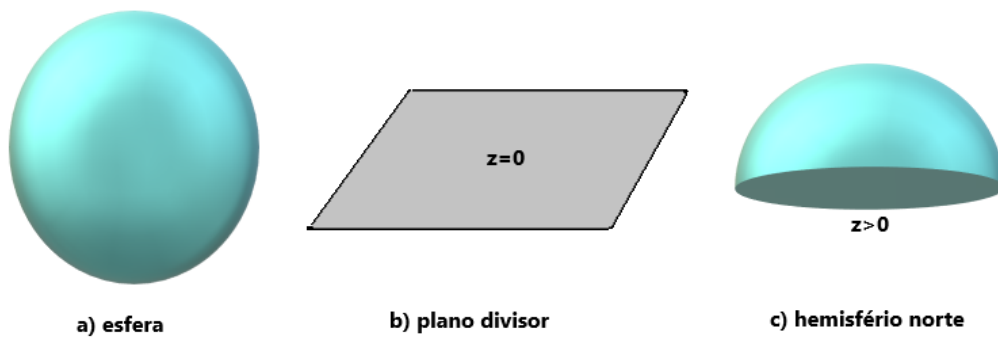


Figura 4.4: Semi-espço de um hemisfério construído por CSG: esfera centrada na origem cortada pelo plano $z=0$.

Uma superfície é definida como o zero de sua equação cartesiana. Um semi-espço é a região de pontos positivos ou negativos da equação de sua superfície (Equação 4.3). Vejamos: para construir um hemisfério da Figura 4.4c, tomemos uma esfera de raio igual a 1, Figura 4.4a, centrada na origem e definida pela Equação 4.2 e um plano divisor perpendicular a z , Figura 4.4b. Então, por definição, o semi-espço negativo é dado por $f(x, y, z) < 0$, e o semi-espço positivo é dado por $f(x, y, z) > 0$. Ou seja, convencionamos que o semi-espço negativo é o conjunto de pontos dentro da superfície, e o semi-espço positivo é o conjunto de pontos fora da superfície.

$$f(x, y, z) = x^2 + y^2 + z^2 - 1 = 0. \quad (4.3)$$

De posse destes conceitos, se quisermos definir uma esfera é necessário criar um objeto da classe `openmc.Sphere` e depois, e atribuir os operadores unários `-` e `+` aos semi-espacos negativo (dentro) e positivo (fora), respectivamente.

```
esfera = openmc.Sphere(r = 1.0)
dentro = -esfera
fora = +esfera
```

Quando `r` é maior que o raio da esfera, o ponto está fora do volume da esfera; quando é menor, o ponto está dentro. Então, se a esfera é o sistema que será simulado, durante a simulação o código identificará as duas regiões e fará as contagens dentro do volume ($r < 1$).

O hemisfério norte é construído com o operador booleano `&` que delimita a região abaixo da esfera (`-esfera`) e acima do plano divisor $z=0$ (`+plano_z`), criado como objeto da classe `openmc.Plane`. Este é um exemplo de como as regiões das células serão criadas.

```
plano_z = openmc.Plane(z = 0.0)
hemisferio_norte = -esfera & +plano_z
```

Um pino do reator proposto é apresentado na Figura 4.5 gerada pelo OpenMC utilizando o conceito de célula, que é um objeto da classe `Cell`. O método `fill`, que preenche a célula com um dos materiais do arquivo `materials.xml`, e o método `region`, que delimita uma região (volume) que será preenchido com o material do método `fill`, são responsáveis por gerar as regiões de combustível (amarelo), gap (branco), cladding (vermelho) e água (azul), com os correspondentes materiais, cada um destacado por uma cor. As células que geram as regiões são descritas a seguir:

```
fuel_cell = openmc.Cell(cell_id=1)
fuel_cell.fill = fuel
fuel_cell.region = -PinoPuro
```

```

gap_cell = openmc.Cell(cell_id=2)
gap_cell.fill = air
gap_cell.region = +PinoPuro & -clad_inner_radius

clad_cell = openmc.Cell(cell_id=3)
clad_cell.fill = zircaloy
clad_cell.region = +clad_inner_radius & -clad_outer_radius

water_cell = openmc.Cell(cell_id=4)
water_cell.fill = water
water_cell.region = +clad_outer_radius

```

Nas 4 células acima, nota-se o papel do operador booleano `&` delimitando as regiões de gap e cladding, e os operadores unários `+` e `-`, que definem se uma região é para fora da superfície (positiva) ou para dentro (negativa). As superfícies que geram as regiões com os operadores citados estão indicadas na Figura 4.5.

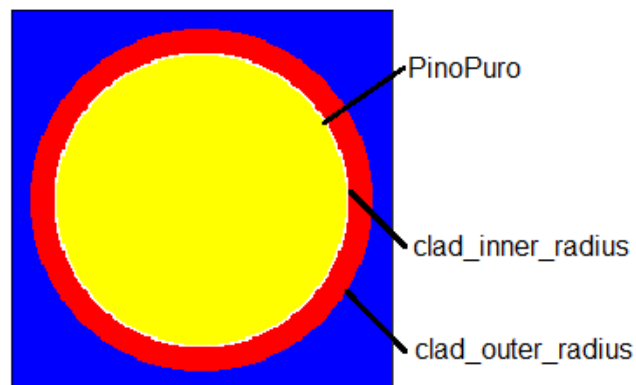


Figura 4.5: Universo Pino gerado com OpenMC; as superfícies usadas para construir as regiões das células.

As células servem para criar outro objeto: o **universo**. Da mesma forma que o conceito de célula é presente em outros códigos de Monte Carlo (como MCNP, por exemplo), o conceito de universo também é conhecido (ROMANO et al., 2015). Um pino, como da Figura 4.5, é uma coleção de células que é utilizada para representar outros pinos

idênticos distribuídos pelo reator. O universo **Pino**, da classe `openmc.Universe`, é uma maneira de criar uma parte do sistema (no caso, um pino combustível), que é referenciada toda vez que numa região do sistema exista pino combustível. A Figura 4.5 é gerada pelo plotagem do universo Pino usando o método `plot` da classe `Universe`:

```
Pino = openmc.Universe(universe_id=1, cells=[fuel, gap, cladding, water])
Pino.plot(width = (1.0, 1.0)) → tamanho da imagem: 1cm x 1cm
```

A geometria do Pino é criada como objeto da classe `openmc.Geometry`, utilizando como atributo o universo Pino. O arquivo `geometry.xml` é finalmente criado com o método `export_to_xml()` e implementado como:

```
geometry = openmc.Geometry(Pino)
geometry.export_to_xml()
```

4.2.3 Arquivo de entrada: `settings.xml`

Os arquivos de entrada tem uma ordem para serem criados: primeiro o arquivo `materials.xml`, depois `geometry.xml`, e, por último, o arquivo de parâmetros de simulação, `settings.xml`, escrito com a classe `openmc.Settings`. Existem vários tipos de parâmetros disponíveis pela classe, no entanto a maioria são opcionais e para avaliações muito específicas. Para o reator proposto serão necessários: número de partículas a serem simuladas (**particles**), modo de simulação **eigenvalue**, ciclos (**batches**), contagens descartadas (**inactives**), e como a fonte inicial está distribuída.

O modo **eigenvalue** (literalmente: **modo autovalor**) diz respeito à simulação onde a fonte de nêutrons inclui o próprio material que será fissionado. Por isso este modo é usado na simulação de reatores nucleares, um sistema que contém material físsil (como o Pino, por exemplo). A razão disso, do ponto de vista da física de reatores, é a equação de transporte de nêutrons se transformar numa equação de autovalor, uma vez que a própria fonte de nêutrons vai depender do fluxo de nêutrons devido à presença do material físsil,

OPENMC-Contributors (2020). Este é o modo setado pelo atributo `Settings.run_mode` em simulações para avaliação de criticidade, que é o caso do reator proposto. Neste modo o atributo `Settings.source` especifica a fonte de início que é apenas usada para provocar a primeira geração de fissão (o “start” da simulação).

O modo eigenvalue está relacionado **Método de Gerações Sucessivas**, proposto por LIEBEROTH (1968). Segundo este modelo, ao invés de rastrear a trajetória dos nêutrons nascidos das fissão durante a simulação, os locais onde a reação de fissão acontece, bem como a direção e energia do nêutron são registrados, armazenados e usados na geração seguinte. Esses dados são armazenado em vetores, para que posteriormente, possam ser manipulados pela biblioteca NumPy do Python. Esses vetores são chamados **Bancos de Fissão**. Ao fim de cada geração, há um número finito N de fontes de fissão para a próxima geração, que servirão para amostragem de M locais de fissão armazenados, garantindo que a população de nêutrons não cresça exponencialmente.

O número total de partículas simuladas é setado no atributo `Settings.particles`, que, para o no modo eigenvalue, este número é dividido em ciclos (ou bateladas - batches) através do atributo `Settings.batches`, estabelecendo assim a quantidade de partículas/ciclo que serão simuladas. Por default, um ciclo corresponde a uma geração de fissão, mas através do atributo `Settings.generations_per_batch` esta quantidade pode ser modificada, (OPENMC-Contributors, 2020).

O atributo `Settings.inactive` serve para determinar o número de ciclos que serão descartados antes do início das contagens (`tallies`) e do fator de multiplicação, k .

A fonte de início é um objeto da classe `openmc.Source`, cujo atributo é um outro objeto de uma subclasse que descreve o tipo de fonte, que pode incluir ainda outros atributos a respeito da distribuição espacial (`Source.space`), angular (`Source.angle`) e energética (`Source.energy`) da fonte. Para o caso da simulação do reator proposto, a fonte de início é objeto da subclasse `openmc.stats.Box`, que consiste numa fonte de início uniformemente distribuída em formato de caixa (box), com dimensões do núcleo do reator abrangendo toda área físsil. Se o Pino da Figura 4.5 fosse simulado, por exemplo, no seu arquivo `settings.xml`, a subclasse `openmc.stats.Point`, que consiste numa fonte

pontual na origem dos eixos cartesianos (0, 0, 0), seria utilizada como fonte inicial, como mostrado a seguir:

Um objeto de `openmc.Source` é criado tendo como atributo um outro objeto de `Point`:

```
Fonte inicial pontual
point = openmc.stats.Point((0, 0, 0))
source = openmc.Source(space=point)
```

Criada a fonte pontual, o objeto de `openmc.Settings` é criado em seguida com os seguintes atributos:

```
settings = openmc.Settings()
settings.run_mode= 'eigenvalue' → parâmetro default
settings.source = source
settings.batches = 100 → 100 ciclos com uma geração de fissão contada
settings.inactive = 10 → 10 ciclos descartados
settings.particles = 1000 → 1000 partículas simuladas em cada um dos 100
ciclos
```

Logo após, o arquivo `settings.xml` é criado com o método `export_to_xml()` e implementado como:

```
settings.export_to_xml()
```

Estes arquivos são requisitos mínimos da simulação, isto é, quando o método `openmc.run` é implementado pelo usuário, imediatamente, o sistema lê a variável ambiente onde estão as seções de choque e os arquivos `materials.xml`, `geometry.xml` e `settings.xml`. Além desses três, há outros arquivos com extensão `xml`, mas são opcionais, como o arquivo de contagens, por exemplo, que será discutido a seguir.

4.2.4 Arquivo de entrada: tallies.xml

O arquivo `tallies.xml` é o input que registra quais as quantidades serão contadas: é o arquivo de contagens. A classe `openmc.Tallies` é utilizado para criar o arquivo das contagens para exportá-lo no final. Numa mesma simulação é possível realizar diversas contagens, e cada uma é objeto de `openmc.Tally`, com filtros que podem ser setados. Os filtros dependem das quantidades de interesse: se a fuga (**leakage**) através de uma superfície é a contagem, então um filtro de corrente (**current**) precisa ser setado. Além disso, alguns tipos de contagem necessitam de uma malha (**mesh**) que discretiza o sistema cobrindo-o como uma rede, que é criada como um objeto de `openmc.RegularMesh`. Esta malha, para o caso da fuga, deve estar associada com um filtro de superfície, `openmc.MeshSurface`, e outro de filtro de energia, `openmc.EnergyFilter`, que possui como atributo um intervalos de energia de interesse. Assim, a escolha de um ou mais filtros está diretamente ligada à quantidade que se quer contar.

Alguns exemplos de contagens: fluxo, taxa de reações de fissão, taxa de produção de nêutrons, taxa de absorção etc. Todas as contagens utilizam, por default, o estimador de caminho percorrido (**Track-Length**). Porém, alguns tipos de contagem requerem outros estimadores, como o de colisão (**Colision**) ou absorção (**Absorption**). A escolha do estimador também está diretamente ligada à quantidade que se quer contar. E a troca do estimador é feita trocando-se o atributo da classe.

Um esquema da construção do arquivo `tallies.xml` é mostrado a seguir, destacando quatro passos:

Primeiro passo: Criação do objeto Tallies

```
tallies = openmc.Tallies([ ]) → lista vazia como atributo
```

Segundo passo: Criação da cada contagem como objeto de Tally

```
tally_1 = openmc.Tally(attribute1, attribute2, ...)
```

```
tally_2 = openmc.Tally(attribute1, attribute2, ...)
```

```
⋮
```

```
tally_N = openmc.Tally(attribute1, attribute2, ...)
```

Terceiro passo: o método `append` adiciona as contagens à lista vazia

```
tallies.append(tally_1)
```

```
tallies.append(tally_2)
```

```
⋮
```

```
tallies.append(tally_N)
```

Quarto passo: Criação do arquivo xml

```
tallies.export_to_xml()
```

4.2.5 Arquivos de saída: Outputs

Ao término da simulação o próprio código cria seus arquivos de saída, os chamados **output**:

- `tallies.out`: arquivo ASCII contendo apenas a média e o desvio padrão das contagens pré-definidas;
- `summary.h5`: arquivo HDF5 com a descrição da geometria e da composição;
- `statepoint.#.h5`: arquivo HDF5 com os resultados completos da simulação, incluindo as contagens, o k_{eff} , e a final distribuição da fonte (convergada).

Os outputs `summary.h5` e `statepoint.#.h5` são internos do código e são carregados em variáveis para apenas serem manipulados por meio de vetores no pós processamento. O output `tallies.out` com as médias e os desvios padrão podem ser visualizados usando um comando do Linux (que pode ser executado também pelo Python), o **cat**, que permite a exibição do arquivo txt. Abaixo, o arquivo de saída da simulação do Pino da Figura 4.5:

```
!cat tallies.out
```

>TALLY 1<

Cell 3

U235

Total Reaction Rate	0.726151 +/- 0.00251702
Fission Rate	0.543836 +/- 0.00205084
Absorption Rate	0.652874 +/- 0.002424
(n,gamma)	0.10904 +/- 0.000385793

A extensão `.h5` identifica arquivos tipo HDF5. Este tipo torna possível o gerenciamento de dados grandes e complexos; é o arquivo padrão do Python, permitindo manipular dados em vetores criados pelo módulo NumPy.

O arquivo `statepoint.#.h5` é mais utilizado pois geralmente a tarefa de tratar os resultados de simulação é facilitada quando se trata o arquivo como um objeto de `openmc.StatePoint`, (OPENMC-Contributors, 2020). Os vetores das contagens são utilizados para plotagem de gráficos, construção de tabelas, histogramas, mapas de fissão etc.

5 Modelagem do Reator Proposto

Para construir o reator proposto no código a geometria do sistema precisa ser definida. O reator simulado teve como a base o SMR CAREM 25 (MAGAN et al., 2014), com informações da própria CNEA e das publicações dos órgãos e pesquisadores associados ao projeto (DELMASTRO et al., 2010). A modelagem do reator começará pelos pinos, depois pelos elementos combustíveis, e, por último, pela configuração do núcleo.

5.1 Pinos

Um pino é uma representação do código. Como se todas as pequenas pastilhas da vareta fossem empilhadas num único pino definido por uma superfície cilíndrica. O pino combustível foi modelado tomando como base as dimensões da vareta combustível do CAREM 25. Na Tabela 5.1, as dimensões e os materiais constituintes das varetas combustíveis do CAREM 25 são apresentados.

Principais Dimensões / Materiais	Dados
Comprimento Total - Total Length	1600mm
Comprimento Ativo - Active Length	1400mm
Diâmetro mais externo do Revestimento (Cladding)	9mm
Espessura do Revestimento - (Cladding)	0.625mm
Diâmetro da Pastilha	7.6 mm
Comprimento da Pastilha	8.0mm
Densidade do Combustível	95%
Volume do plenum	6.8cm ³
Material do Revestimento - (Cladding)	Zircaloy - Zry-4
Combustível da Pastilha	UO ₂ / UO ₂ -Gd ₂ O ₃

Tabela 5.1: Dimensões principais da vareta combustível do CAREM 25 (MARKIEWICZ, 2014).

Tendo as dimensões da vareta combustível do CAREM 25 como base, a Figura 5.1 apresenta um corte radial feito com o OpenMC do pino do reator proposto que representa qualquer um dos três tipos vareta combustível. Essas dimensões também servem para delimitar as regiões do pino em : combustível, gap, cladding e meio circundante (refrigerante/moderador). Os materiais que preenchem estas regiões (declarados pelo usuário no arquivo `materials.xml`) servem para as criar **células** que, por sua vez, servem para indicar onde o código deve realizar um certo tipo de contagem: se é fissão, ela será feita apenas na região amarela, onde existe material que este tipo de reação ocorre.

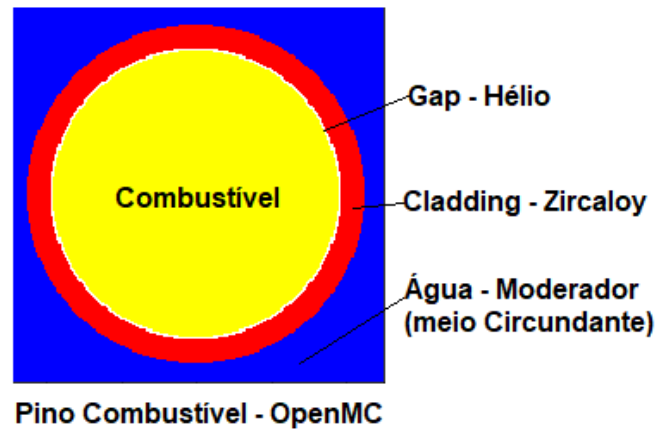


Figura 5.1: Pino Combustível obtido com a classe Plot do OpenMC; em destaque, regiões de combustível, gap, cladding e moderador/refrigerante.

O hélio será usado na simulação como material do gap. A CNEA não informa se este é o material de gap do CAREM 25. Como hélio é usado nas varetas dos reatores PWR como material do gap e o reator modelado é baseado no CAREM 25 (um PWR integrado), ele foi adaptado como material do gap.

Os pinos com veneno seguem a mesma lógica. A Figura 5.2 é um corte ao axial de um pino com veneno queimável mostrando sua configuração heterogênea: UO_2 nas extremidades e $\text{UO}_2\text{-Gd}_2\text{O}_3$ na região intermediária.

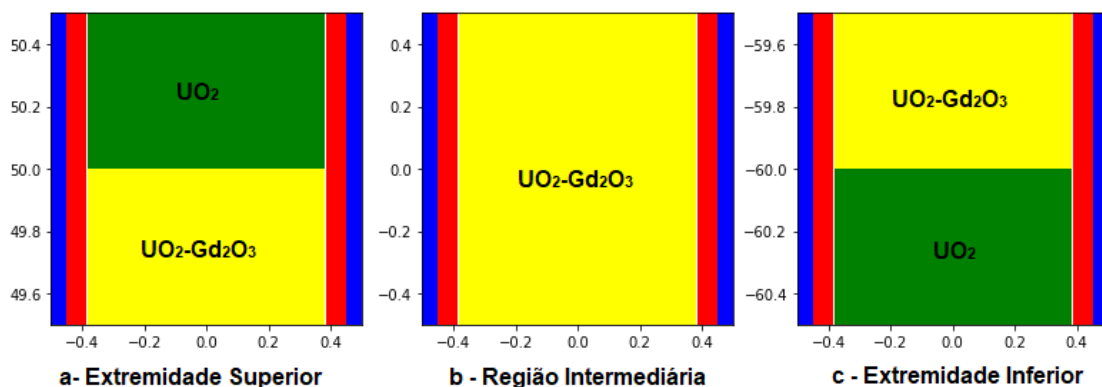


Figura 5.2: Pino com veneno queimável; corte axial com a classe Plot do OpenMC.

Os pinos que representam as barras de controle do reator foram modelados com

base as dimensões das barras do CAREM 25. Na Tabela 5.2, as dimensões e os materiais constituintes das barras de controle do CAREM 25 são apresentados:

Principais Dimensões / Materiais	Dados
Comprimento Total - Total Length	1708mm
Comprimento Ativo - Active Length	1400mm
Diâmetro mais externo do revestimento (Cladding)	8.5mm
Espessura do Revestimento - (Cladding)	0.65mm
Diâmetro da Pastilha	7mm
Comprimento da Pastilha	50mm
Material do Revestimento - (Cladding)	AISI 316L
Material Absorvente	Ag-In-Cd

Tabela 5.2: Dimensões principais das barras de controle do CAREM 25 (MARKIEWICZ, 2014).

Os pinos das barras de controle são apresentados nas vistas axial, Figura 5.3a, destacando a ponta da barra inserida no tubo guia, e radial, Figura 5.3b, na vista de topo, mostrando as regiões do material constituinte do elemento absorvedor, Ag-In-Cd; do gap, água leve; do revestimento (cladding), aço AISI 316L; e do meio circundante (refrigerante/moderador), água leve.

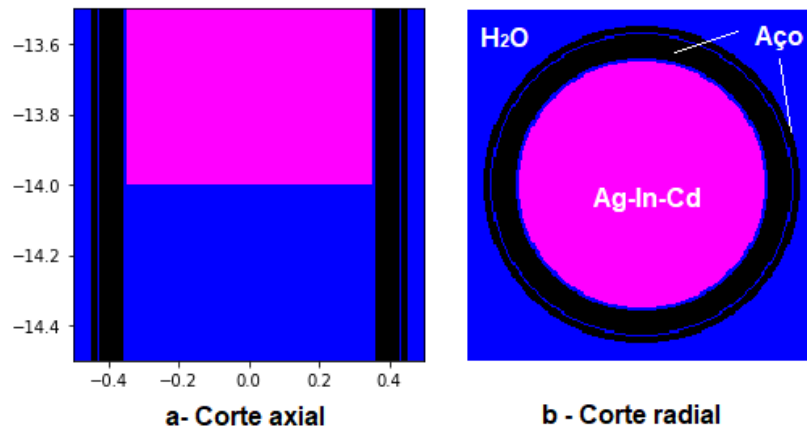


Figura 5.3: Barra de Controle: cortes axial e radial obtidos com a classe Plot do OpenMC.

5.2 Elemento Combustível

Antes de proceder na representação dos elementos, um artifício geométrico precisa ser feito. Na Figura 5.4, um cilindro cuja altura é igual ao comprimento ativo é dividido ao meio pelo plano z de referência ($z=0$), resultando em duas metades simétricas. Desta maneira foram modelados os pinos, e, por consequência, os elementos combustíveis e o núcleo. Esse procedimento é útil para posicionar a fonte inicial na simulação: se é pontual (objeto de `openmc.stats.Point`), basta posicioná-la na origem dos eixos $(0, 0, 0)$; se a fonte inicial é em formato de caixa (objeto de `openmc.stats.Box`), suas dimensões também podem usar este artifício geométrico para posicioná-la na origem dos eixos $(0, 0, 0)$, resultando numa caixa literalmente no “meio” do sistema.

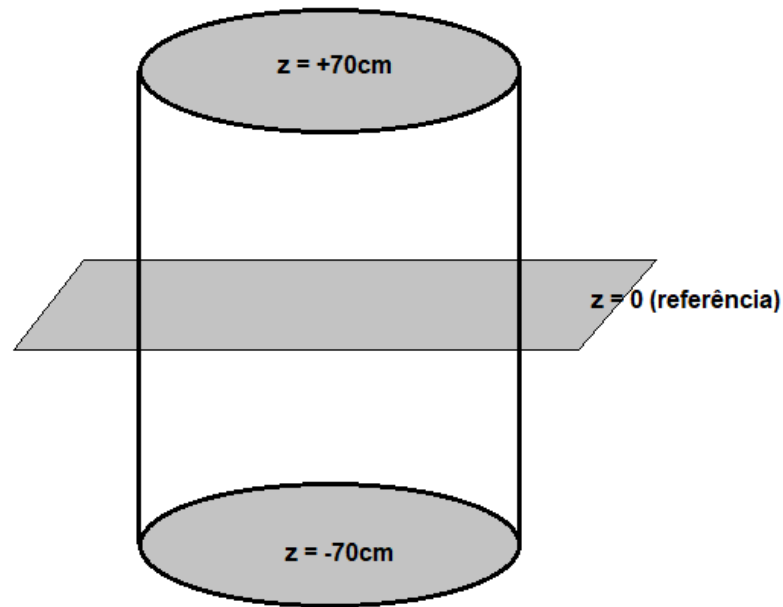


Figura 5.4: Divisão dos componentes por um eixo de referência. Marco (2022).

Esta representação é útil também para inserção das barras de controle, pois estabelece o limite superior como referência para localizar o plano da ponta da barra à medida que ela é inserida/retirada no núcleo em passos fixos, subtraindo o valor do passo (14cm) do limite superior: primeiro passo, em $z=56$; segundo passo, em $z=42$, e assim em diante. A Figura 5.5 apresenta o esquema de inserção de 1 até 10 passos (completamente inserida no tubo guia). A inserção/retirada das barras de controle se resume a modificar o valor da variável z no código para diminuir/aumentar a reatividade do núcleo.

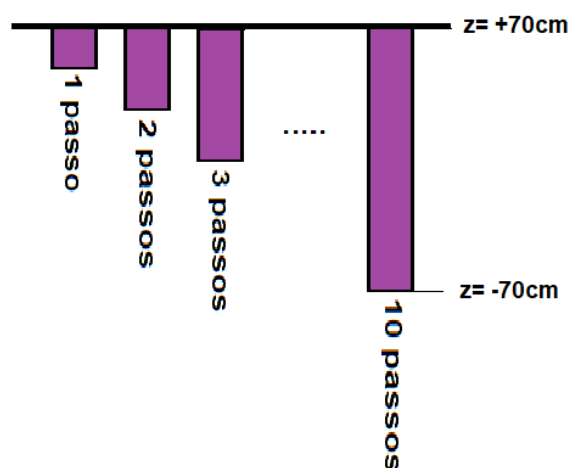


Figura 5.5: Passos de inserção das barras de controle. Marco (2022).

A Tabela 5.3 apresenta os dados principais do elemento combustível do CAREM 25. Baseando-se nesses dados os elementos combustíveis do reator proposto serão modelados. A distância centro a centro, denominada **pitch**, é um parâmetro fixo que serve para delimitar o tamanho do canal por onde passa o refrigerante. O pitch é útil tanto para o OpenMC modelar os ECs individualmente, quanto para cálculos de engenharia de reatores. O mesmo para a distância entre ECs, que serve para a modelagem do núcleo do reator no OpenMC, como será mostrado na seção seguinte.

Dimensões Principais	Dados
Forma	Hexagonal
Comprimento Total (cm)	180
Número de varetas	108
Número de Tubos para Barras de Controle	18
Número de Tubos para Instrumentação	1
Número de Grades Espaçadoras	4
Pitch (cm)	1.38
Distância entre ECs (cm)	16
Peso total do elemento (Kg)	100

Tabela 5.3: Dimensões principais do elemento combustível do CAREM 25 (MARKIEWICZ, 2014).

A seguir serão modelados os tipos de elementos que farão parte do núcleo do reator proposto.

Um elemento com barras de controle é apresentado na Figura 5.6. Os pinos combustíveis são de 1.8% enriquecimento e não há pinos com veneno queimável. Na cor oliva, o pino combustível; na cor branca, o gap de hélio; na cor vermelha, o cladding das varetas, feito de zircaloy; na cor rosa, o pino absorvedor e, na cor preta, o revestimento

dos tubos e das barras, feito de aço AISI 316L. Na Figura 5.6a, o tubo de instrumentação aparece cheio d'água; na Figura 5.6b, as barras de controle são visualizadas na posição de 1 passo de inserção (14 cm - 10% do comprimento ativo), destacado na cota $z=56$, em relação à cota superior ($z=70$), conforme explicado anteriormente.

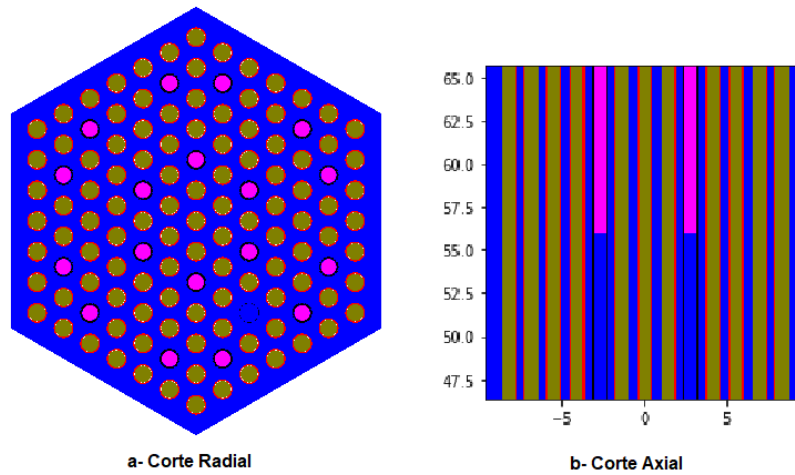
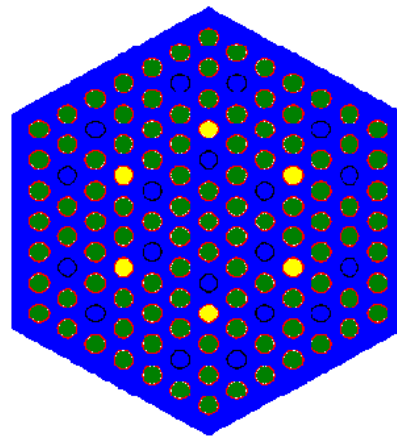


Figura 5.6: Elemento com barras de controle: cortes radial e axial plotados com OpenMC.

Um elemento combustível que possui seis pinos contendo Gd_2O_3 é apresentado na Figura 5.7. Este tipo de pino pode ser visualizado dentro do EC. Na cor verde, as regiões de combustível puro (3.1% de enriquecimento); na cor amarela, as regiões com Gd_2O_3 . As cotas delimitando onde começam as regiões com Gd_2O_3 são destacadas nas Figuras 5.8a e 5.8b (10cm acima da base, em relação à cota inferior, $z=-70$; e 20cm abaixo do topo do comprimento ativo, em relação à cota superior, $z=70$). Os tubos guia aparecem cheios d'água pois este elemento não possui banco de barras de controle.



Corte Radial

Figura 5.7: Elemento contendo seis pinos com veneno queimável plotado com OpenMC.

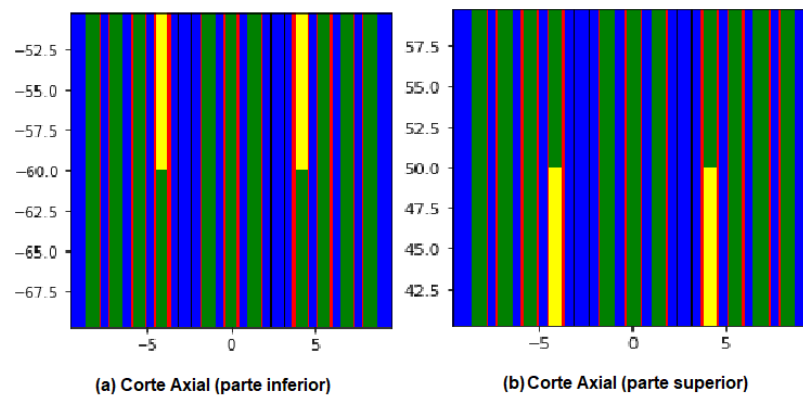


Figura 5.8: Elemento com seis pinos com veneno queimável - corte Axial.

Elementos combustíveis que possuem 6 pinos com veneno queimável podem ter bancos de 12, 18 ou nenhuma barra, dependendo do sistema de controle; elementos com 12 pinos contendo veneno queimável não possuem bancos de barra; outros não contêm nem pinos com veneno queimável nem barras. A seguir, nas Figuras 5.9 e 5.10, estão apresentadas as configurações desses elementos com o OpenMC.

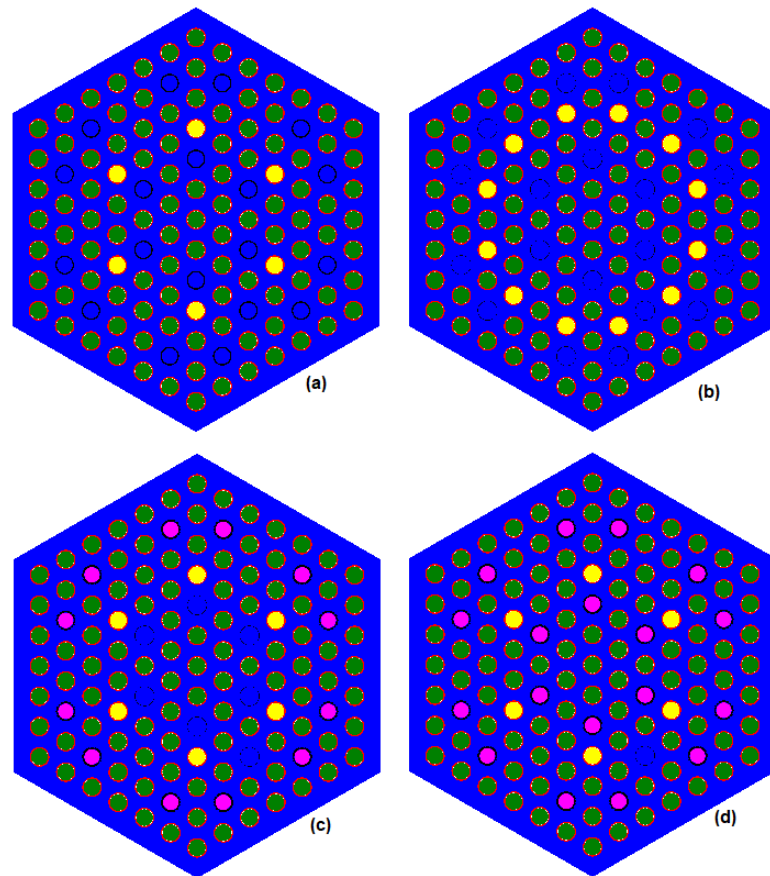


Figura 5.9: Configurações de elementos combustíveis: **a**: 6 pinos contendo veneno queimável; **b**: 12 pinos contendo veneno queimável; **c**: 6 pinos contendo veneno queimável e 12 barras de controle; **d**: 6 pinos contendo veneno queimável e 18 barras de controle.

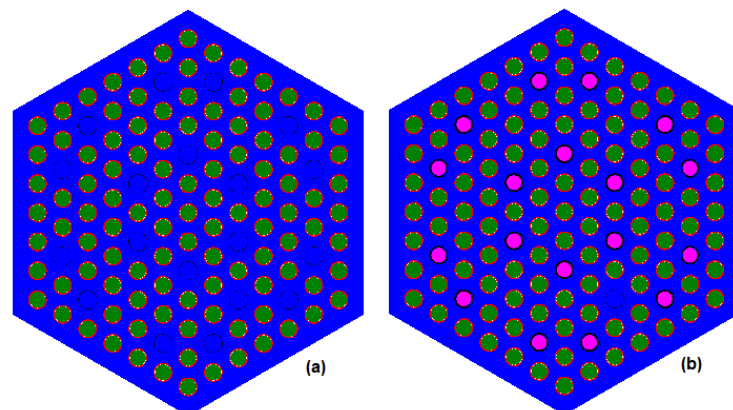


Figura 5.10: Configurações de elementos combustíveis: **a**: sem nenhuma barra de controle; **b**: contendo 18 barras de controle.

5.3 Núcleo

Antes da modelagem do núcleo do reator proposto é necessário estabelecer a distribuição dos elementos combustíveis no núcleo, isto é, como serão as zonas de enriquecimentos e onde serão posicionados os elementos com veneno queimável e com bancos de barras de controle. Como primeiro passo, a organização dos elementos combustíveis precisa ser estabelecida, de modo a facilmente identificá-lo no núcleo, como mostra a Figura 5.11, onde todos os 61 elementos do núcleo estão numerados.

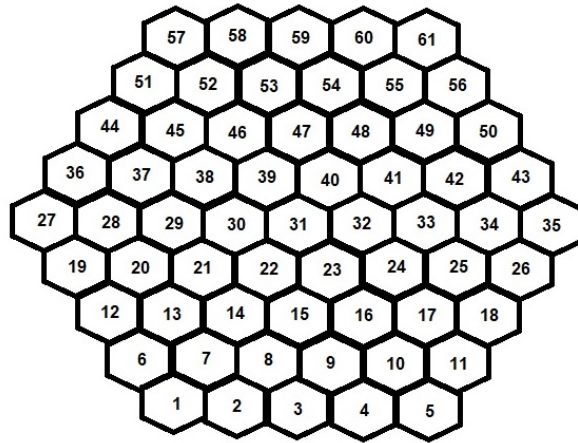


Figura 5.11: Numeração dos elementos combustíveis na modelagem do núcleo do reator. Marco (2022).

O esquema da distribuição de enriquecimentos e varetas com veneno queimável do núcleo do reator é apresentado na Figura 5.12, proposto por TASHAKOR et al. (2017) baseado em (VILLARINO et al., 2012) para o CAREM 25. Nele, estão destacadas três zonas de enriquecimento sem veneno queimável circundadas por elementos com 6 varetas de veneno queimável e outros com 12 em posições simetricamente dispostas; apenas o elemento central possui menor enriquecimento, destacado em vermelho.

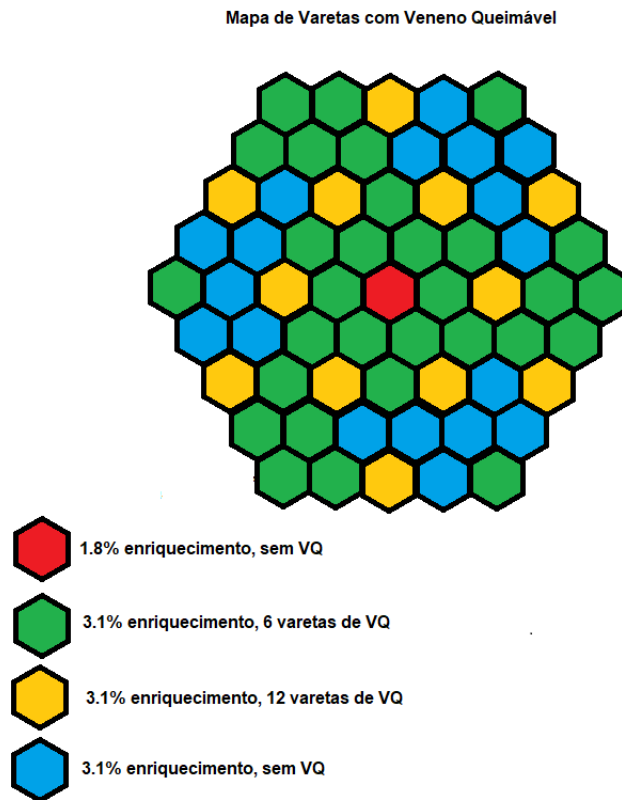


Figura 5.12: Configuração do núcleo do CAREM 25: tipos de enriquecimento/elementos com varetas contendo Gd_2O_3 . TASHAKOR et al. (2017) baseado em (VILLARINO et al., 2012).

A reatividade do núcleo é controlada pelo uso de Gd_2O_3 como veneno queimável em varetas específicas e com elementos absorventes móveis dos Sistemas de Controle e Ajuste. Gd_2O_3 é sinterizado junto com UO_2 na proporção de 7.5% em peso de Gd_2O_3 para achatar a distribuição de potência ao longo do ciclo (VILLARINO et al., 2012).

Os sistemas de controle estão divididos em vários bancos. O primeiro deles, o **SAC**, Sistema de Ajuste e Controle, é utilizado para manter e ajustar a potência do reator e consiste em grupos de 12 ou 18 barras que são inseridas no núcleo pelos tubos guia. A Figura 5.13 mostra 25 elementos combustíveis com bancos de barra de controle. O SAC é subdividido em outros dois: o (**Sistema de Controle - SC**), que atua em situações de emergência ou na operação normal do reator. Esta duplicidade de uso está relacionada ao fato de que o SC está localizado no elemento central onde o fluxo é maior, e por isso atua nos dois casos; o (**Sistema de Ajuste - SA**) consiste em bancos de 12 ou 18 barras

para remover excesso de reatividade. Incorporado a este grupo, está o **RES - Rapid Extension System**, o Sistema de Extinção Rápida, presente em apenas 6 elementos combustíveis (MAGAN et al., 2011). Este sistema é especial porque age somente em situações de emergência ou transiente, e junto com os sistemas passivos de segurança garantem o pronto desligamento do reator.

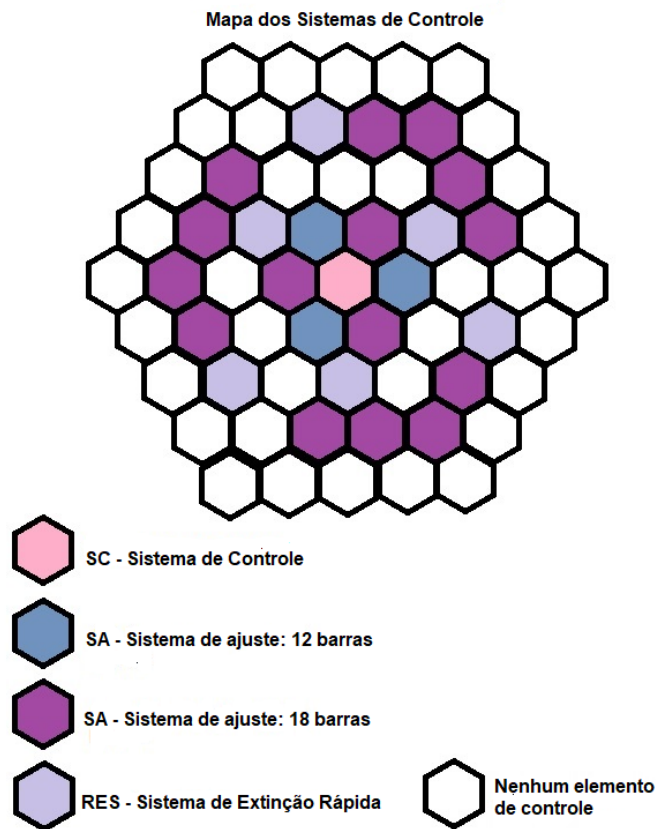


Figura 5.13: Sistemas de Controle. MAGAN et al. (2011) baseado de (VILLARINO et al., 2012).

De posse dos mapeamentos de elementos de controle, da distribuição de enriquecimentos e da numeração, a seguir será plotado o núcleo do reator com a composição final e circundado pelos componentes barrel e, mais externamente, pelo vaso de pressão. Na simulação, ambos serão feitos do mesmo material, AISI 316. A Tabela 5.4 contém as dimensões principais e os materiais do barrel e do vaso de pressão do CAREM 25. Estes dados completam as informações necessárias para a modelagem do núcleo do reator proposto.

Dimensões Principais / Materiais	Dados
Barrel - diâmetro interno	75mm
Barrel - diâmetro externo	77.5mm
Vaso de pressão - diâmetro interno	155mm
Vaso de pressão - diâmetro externo	160mm
Material do Vaso de pressão	AISI 316L
Material do Barrel	AISI 316L

Tabela 5.4: Dimensões principais do barrel e vaso de pressão (CNEA, 2017).

Na Figura 5.14 corte mostra o núcleo do reator proposto baseado no CAREM 25, com os 61 elementos combustíveis, o barrel e o vaso de pressão modelados no OpenMC.

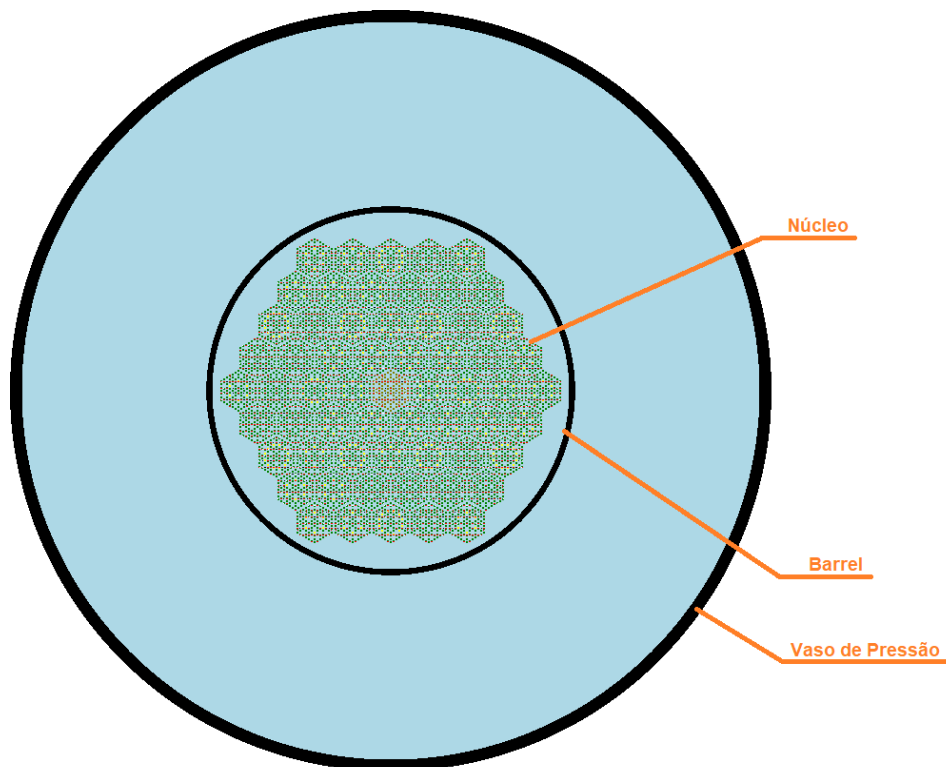


Figura 5.14: Núcleo, barrel e vaso de pressão plotados com OpenMC.

Ampliações da Figura 5.14 são apresentadas nas Figuras 5.15 e 5.16. Nesta última, ao redor do elemento central, com os pinos combustíveis de menor enriquecimento (1.8%) destacados em laranja, há elementos onde podem ser visualizados os pinos com veneno queimável, em amarelo, as barras de controle, em rosa, pinos puros de maior enriquecimento (3.1%), em verde, e os tubos guia vazios (buracos d'água).

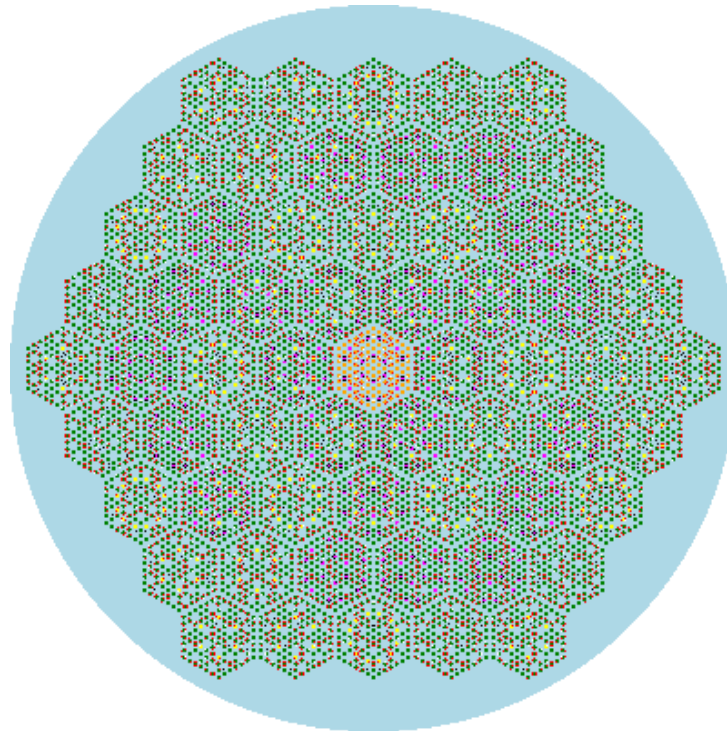


Figura 5.15: Núcleo do reator proposto que será simulado no OpenMC.

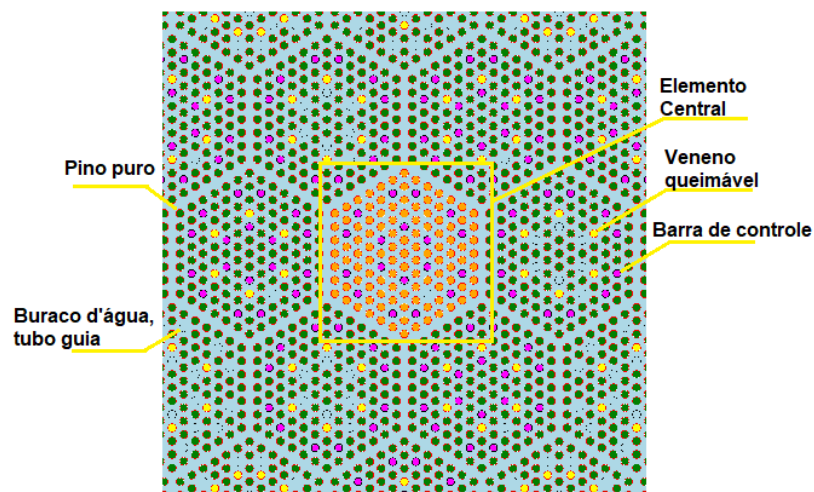


Figura 5.16: Detalhe do núcleo do reator proposto mostrando o elemento central.

6 Benchmark BEAVRS

Como forma de avaliar o funcionamento do código, também será simulado um modelo conhecido, o benchmark BEAVRS. Este modelo consiste num reator PWR convencional, que, por tratar-se de um tipo já simulado com outros códigos estabelecidos e ter resultados conhecidos quanto à criticidade, permitirá verificar se corrobora ou não comportamentos já conhecidos deste tipo de reator. Se corrobora, o código prova-se apto para a simulação do reator proposto.

O benchmark BEAVRS (**Benchmark for Evaluation and Validation of Reactor Simulations**) foi também desenvolvido pelo **Massachusetts Institute of Technology (MIT)**, versão lançada em Dezembro/2020 e elaborada pelo Grupo de Física Computacional de Reatores do MIT. O benchmark BEAVRS consiste num PWR/Westinghouse com dois ciclos de carregamento, refrigerado e moderado com água borada, com descrição detalhada dos elementos combustíveis, das varetas de veneno queimável, dos padrão de carregamento no núcleo e de outros componentes presentes dentro do vaso de pressão. Este benchmark pode ser utilizado para validação de transporte de nêutrons, termo-hidráulica e depleção isotópica do combustível (HORELIK et al., 2020).

São disponíveis as configurações das zonas de enriquecimentos e dos elementos com varetas de veneno queimável para dois ciclos de carregamento: ciclo 1 e ciclo 2. Um corte do núcleo do BEAVRS é apresentado na Figura 6.1. Nela, as regiões de carregamento 1, 2 e 3 para o ciclo 1 aparecem destacadas em vermelho, amarelo, e azul, respectivamente; a Tabela 6.1 apresenta o carregamento das regiões para os dois ciclos. A simulação do benchmark será feita com o configuração do ciclo 1.

Núcleo	
Nº elementos combustíveis	193
Carregamento	(%) em peso de U^{235}
Região 1 (ciclo 1)	1.6
Região 2 (ciclo1)	2.4
Região 3 (ciclo1)	3.1
Região 4A (ciclo 2)	3.2
Região 4B (ciclo 2)	3.4
Quantidade de metal pesado (ciclo 1)	81.8 MT

Tabela 6.1: Regiões de carregamento do núcleo do benchmark para os ciclos 1 e 2 (HORELIK et al., 2020).

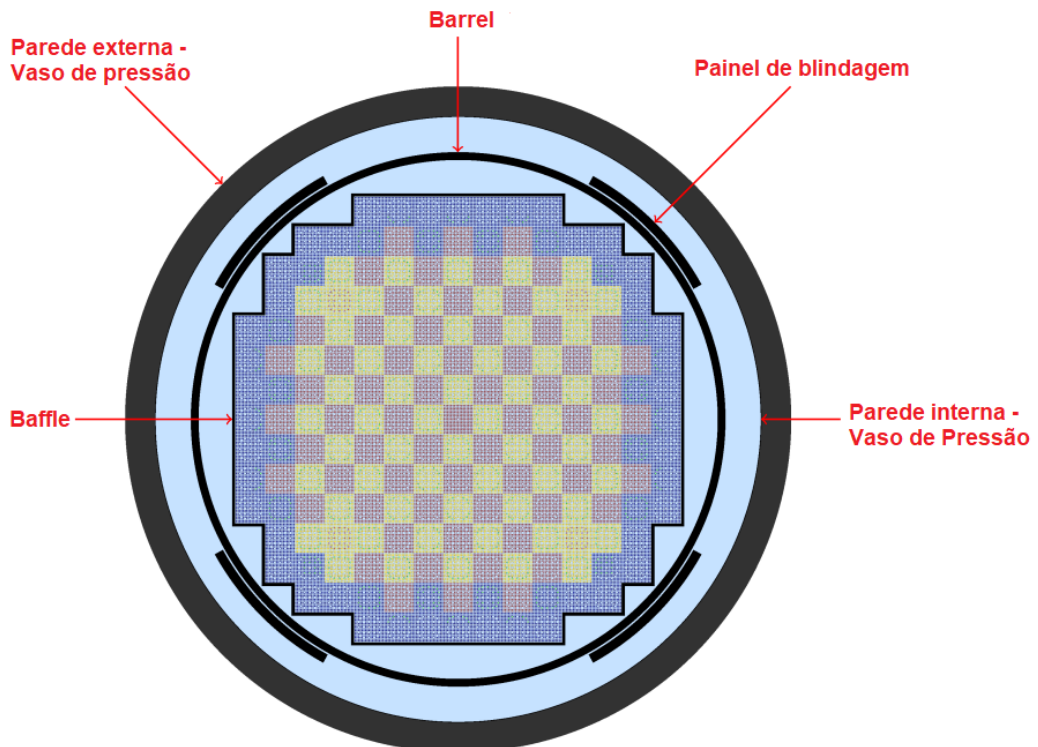


Figura 6.1: Seção do núcleo do benchmark com regiões de enriquecimento. Destaque para os materiais estruturais do núcleo na cor preta; em azul claro, a água; e em vermelho, amarelo e azul escuro as regiões de carregamento do ciclo 1, com 1.6, 2.4 e 3.1% em peso de U^{235} , respectivamente (HORELIK et al., 2020).

O núcleo do benchmark BEAVRS será modelado com a mesma ordem hierárquica que o reator proposto: começando pelos pinos, depois os elementos combustíveis, e, por fim, o núcleo. Pela similaridade dos pinos do BEAVRS com o do reator proposto, só os elementos combustíveis serão descritos, com os pinos podendo ser visualizados dentro dos mesmos. O núcleo do reator será plotado pelo OpenMC com as especificações dos materiais estruturais do núcleo, dos sistemas de controle e da localização dos canais de instrumentação conforme a documentação do BEAVRS.

6.1 Elemento Combustível

O núcleo do BEAVRS possui 193 elementos combustíveis, de arranjo retangular 17 x 17 com as configurações seguintes:

- todos contendo 24 tubos guia para inserção de veneno queimável ou barras de controle;
- contendo nenhuma, 6, 12, 15, 16 ou 20 varetas de veneno queimável;
- contendo nenhum ou um dos bancos de barras de controle: A, B, C, D, S_A , S_B , S_C , S_D ou S_E ;
- contendo ou não tubo de instrumentação (inserido no tubo guia central de cada EC);

A Tabela 6.2 contém os dados principais do elemento combustível, incluindo o material estrutural. As Figuras 6.2 e 6.3 apresentam as distribuições das varetas de veneno queimável por elemento combustível no núcleo e dos sistemas de controle, respectivamente. Em ambas, os elementos aparecem identificados por letra e número. Da mesma forma que no reator proposto, esta numeração será utilizada para identificação dos elementos combustíveis na implementação no OpenMC.

Elemento Combustível

pitch (cm)	1.25984
distância entre ECs (cm)	21.50364
arranjo	17 x 17
Nº varetas combustíveis	264
Nº de posições de tubos guia	24
Nº de posições de tubos de instrumentação	1
comprimento ativo (cm)	365.76
Nº grades espaçadoras	8
material - grades espaçadoras Superior / Inferior	Inconel718
material - grades espaçadoras intermediárias e camisa	Zircaloy
material das camisas Superior / Inferior	Aço Inoxidável 304

Tabela 6.2: Dados principais do elemento combustível do BEAVRS (HORELIK et al., 2020).

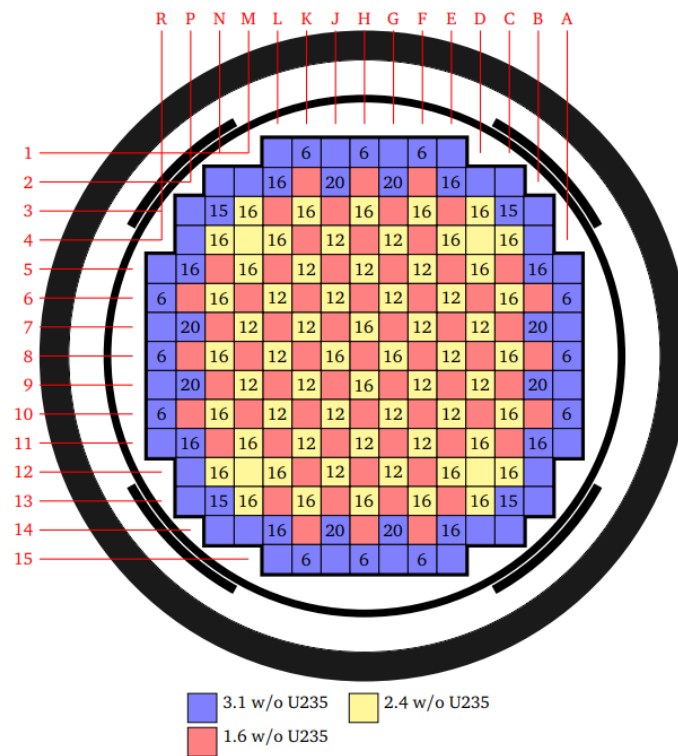


Figura 6.2: Distribuição das varetas de veneno queimável por elemento combustível no núcleo (HORELIK et al., 2020).

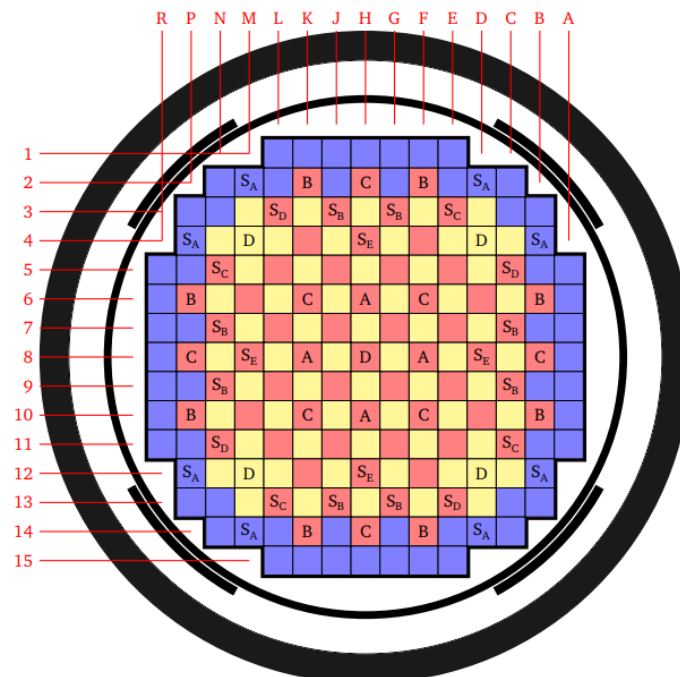


Figura 6.3: Sistemas de Controle: localização dos bancos de barras de controle (HORELIK et al., 2020).

As diferentes configurações do elemento combustível são resultantes do grau de enriquecimento, da inserção de veneno queimável e/ou barras de controle através dos tubos guia, e da presença ou não de tubo de instrumentação, como mostrado nos mapas das Figuras 6.2 e 6.3.

A seguir, serão plotados com o OpenMC as diferentes configurações dos elementos combustíveis.

A Figura 6.4a mostra um elemento combustível com pinos de 1.6% de enriquecimento na cor amarela, com revestimento de Zircaloy-4, na cor vermelha; 24 tubos guia vazios; tubo de instrumentação ao centro, na cor cinza. A Figura 6.4b mostra uma ampliação do corte na altura das grades espaçadoras. Na Figura 6.4c, um corte longitudinal detalha a configuração ao longo do comprimento ativo. O tubo de instrumentação é visto ao centro nas três figuras.

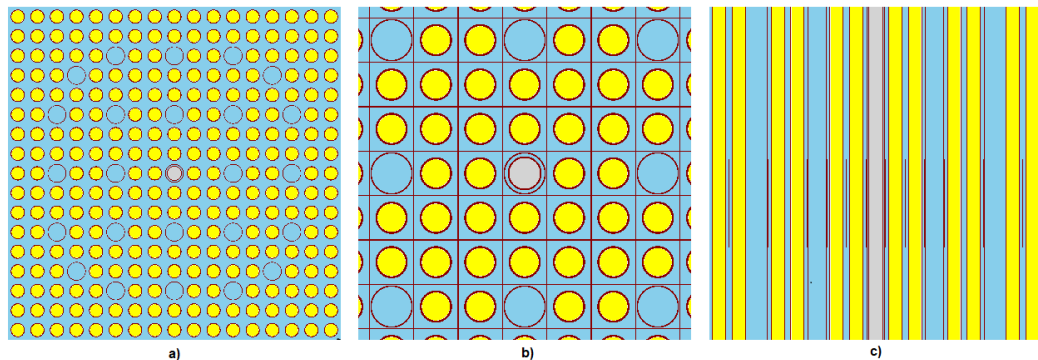


Figura 6.4: Elemento combustível 1.6% de enriquecimento: a) corte transversal; b) zoom; c) corte longitudinal.

A Figura 6.5 mostra, no sentido horário, o corte longitudinal do plenum acima da região do comprimento ativo, seguido de um pino de Zircaloy-4 (na cor vermelha) que delimita o fim do pino; a parte inferior do comprimento ativo, também seguida de um pino de Zircaloy-4 e da região do bocal, na cor cinza escuro; e os cortes transversais dos pinos de Zircaloy-4 e do plenum.

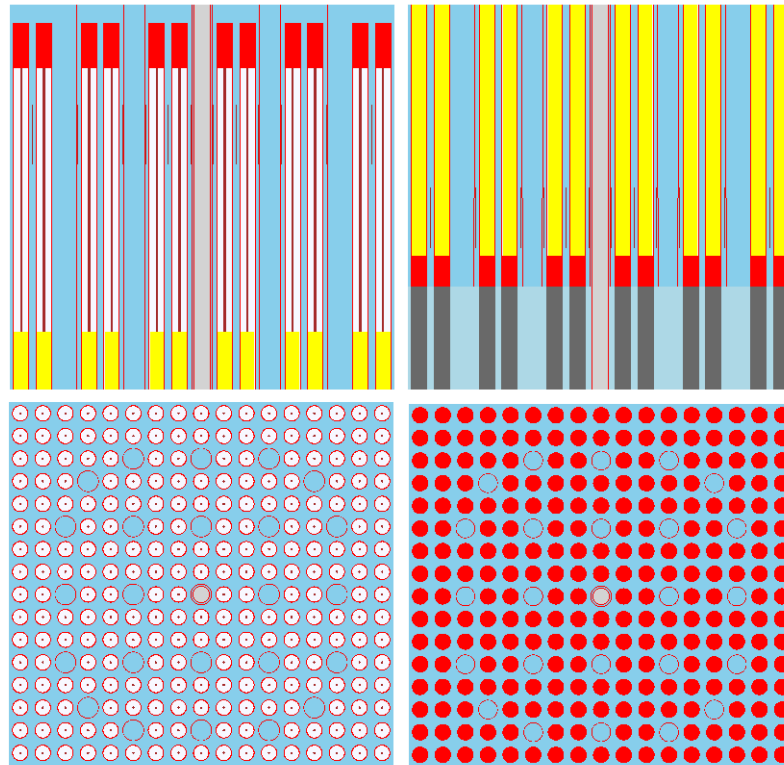


Figura 6.5: Elemento combustível 1.6% de enriquecimento: cortes longitudinal e transversal.

O corte transversal do bocal inferior deste elemento combustível é apresentado na Figura 6.6, destacando as posições dos pinos no bocal.

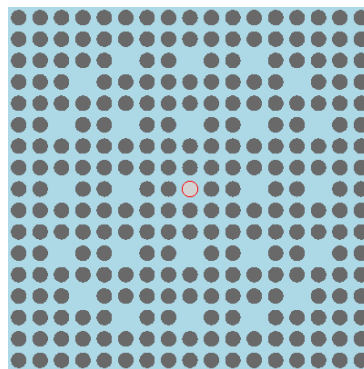


Figura 6.6: Elemento combustível 1.6% de enriquecimento: bocal inferior.

O veneno queimável não possui Gd_2O_3 , nem é misturado ao combustível; ao invés disso, o veneno queimável está na forma de varetas inseridas pelos tubos guia contendo silicato de boro (material na forma vítrea) 12.5% de B_2O_3 , localizado a alguns centíme-

tros acima do início e abaixo do topo do comprimento ativo. A Figura 6.7, apresentam elementos combustíveis com 6, 12, 16 e 20 pinos de veneno queimável.

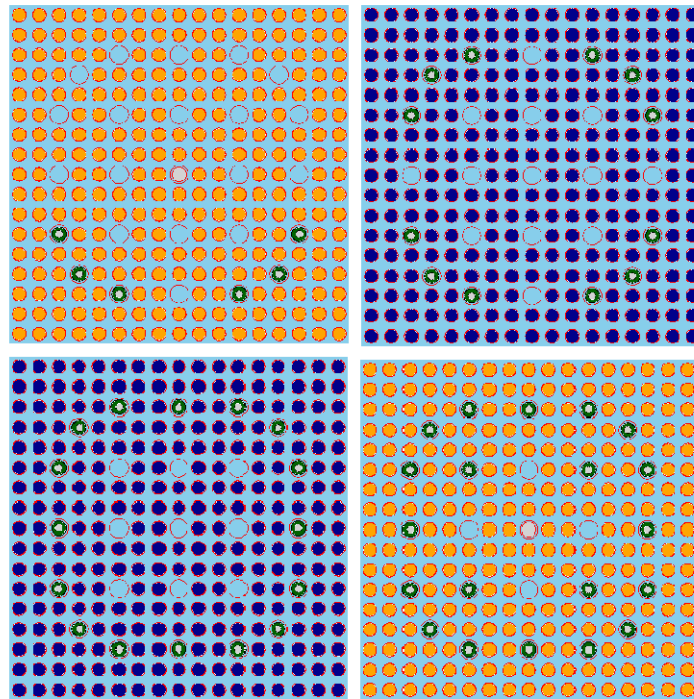


Figura 6.7: Elementos combustíveis com veneno queimável: 6, 12, 16 e 20 pinos de silicato de boro do benchmark.

Apenas 4 elementos possuem 15 pinos de veneno queimável, vide Figura 6.2. A disposição desses pinos não é igual para todos: os elementos N3, N13, C3 e C13, tem esses pinos arranjados em direção ao centro do núcleo, como mostrado na Figura 6.8 para o caso do elemento C3. Os elementos com 6 pinos de veneno queimável também estão orientados desta maneira, como já mostrado na Figura 6.7.

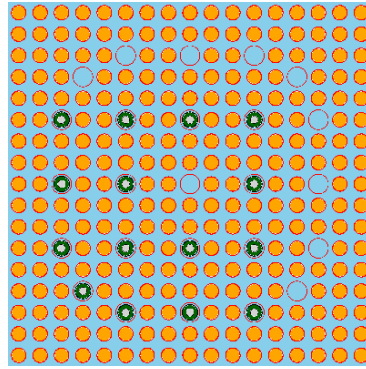


Figura 6.8: Elemento combustível C3 com pinos apontando para o centro do núcleo do benchmark.

Na Figura 6.9, um corte longitudinal mostra a configuração dos pinos de veneno queimável.

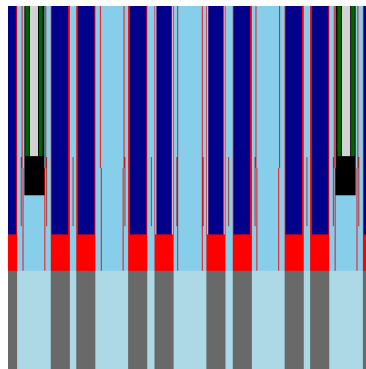


Figura 6.9: Corte longitudinal em um elemento com veneno queimável.

As barras de controle possuem dois tipos de material absorvedor: B_4C e Ag-In-Cd. O sistema de controle D é visualizado na Figura 6.10a. Ao longo do comprimento ativo a região de controle é heterogênea, com o pino de B_4C situado acima da pino de Ag-In-Cd, como na Figura 6.10b. Em destaque, o B_4C na cor verde, e Ag-In-Cd, na cor rosa. O número de passos determina se um ou os dois materiais serão responsáveis por absorver os nêutrons. O passo de inserção equivale a 1.58193cm, com inserção total consistindo em 228 passos, delimitando a região ativa das barras de controle em 360.68cm, um pouco menor que o comprimento ativo (365.76cm). Os cortes foram feitos onde os dois materiais estão na região ativa.

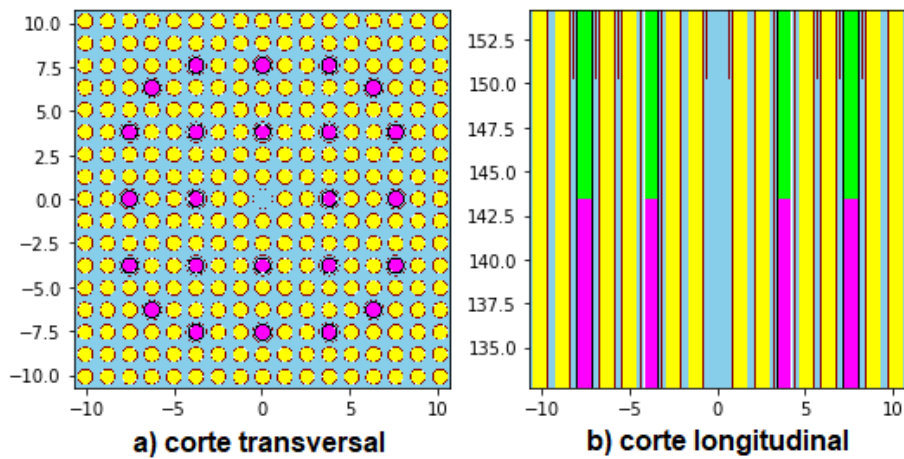


Figura 6.10: Elemento combustível com Sistema de Controle D.

As barras de controle são elementos absorvedores móveis que pertencem aos Sistemas de Controle. Pinos de veneno queimável são elementos estáticos que também pertencem a esses sistemas. A Tabela 6.3 apresenta os aspectos mais importantes dos Sistemas de controle do BEAVRS.

Controle da reação em cadeia

material - barra de controle (região superior)	B_4C
material - barra de controle (região inferior)	Ag-In-Cd
N de bancos de barras de controle	57
material de veneno queimável	Silicato de boro (12.5% em peso de B_2O_3)
Nº de varetas de veneno queimável no núcleo	1266

Tabela 6.3: Controle da reação em cadeia do benchmark (HORELIK et al., 2020).

Os bancos de barras de controle são identificados pelas letras A, B, C e D, cada um constituindo um sistema de controle, conforme mostrado na Figura 6.3. Como no CAREM 25, quando acionados, as aranhas inserem os bancos através dos tubos guia de um só vez, como se fossem uma unidade. Os bancos A, B, C e D são inseridos na ordem D, C, B e A. Os sistemas de emergência do BEAVRS são identificados por S_A , S_B , S_C , S_D e S_E e são acionados apenas para desligar o reator, constituindo assim o Sistema de Desligamento Rápido.

Os sistemas de controle, além dos bancos de controle, conta com o boro diluído em partes por milhão (ppm) no refrigerante, na forma de ácido bórico (H_3BO_3). A concentração inicial na simulação do benchmark foi de 600ppm.

Este tipo de controle é típico de reatores PWRs e chama-se **Controle Fino**. A alta seção de choque de absorção do boro permitem ajustes mais finos da reatividade do núcleo. O controle da quantidade de boro diluído, a purificação do refrigerante, e a homogeneização toda vez que o boro é diluído, fazem parte da operação de reatores que utilizam controle fino. Porém, é necessário observar que este tipo de controle não é tão rápido quanto as barras de controle, pois as etapas de diluição e homogeneização do boro possuem tempos de resposta muito grandes par picos de potência ou transientes (LAMARSH and BARATTA, 2001, pg.363).

6.2 Núcleo

Como mostrado na Figura 6.1 o núcleo também é circundado por componentes estruturais: baffle, barrel, quatro painéis de blindagem localizados em sua periferia e o vaso de pressão (Reactor Pressure Vessel - RPV), com paredes interna e externa. A Tabela 6.4 contém as dimensões destes componentes estruturais.

Componentes Estruturais do núcleo

Espessura - Baffle	2.22250 cm
Gap - Baffle	0.1627 cm
Material - Baffle	Aço Inoxidável 304
Diâmetro interno - Barrel	187.960 cm
Diâmetro externo - Barrel	193.675 cm
Material - Barrel	Aço Inoxidável 304
Diâmetro interno - Painel de blindagem	194.840 cm
Diâmetro externo - Painel de blindagem	201.630 cm
Material - Painel de blindagem	Aço Inoxidável 304
Orientação - Painel de blindagem	32° nas marcas de 45° do reator
Diâmetro interno - (Parede interna)	219.150 cm
Diâmetro externo - (Parede interna)	219.710 cm
Material - (Parede interna)	Aço Inoxidável 304
Diâmetro interno - (Parede externa)	219.710 cm
Diâmetro externo - (Parede externa)	241.300 cm
Material - (Parede externa)	Aço carbono 508

Tabela 6.4: Dimensões dos componentes estruturais do núcleo do benchmark (HORELIK et al., 2020).

A Figura 6.11 é uma seção do núcleo plotada com o OpenMC, com os 193 elementos, juntamente com os componentes estruturais baffle, barrel, painéis de blindagem e vaso de pressão. O núcleo do BEAVRS foi modelado de acordo com a documenta-

ção disponível e para garantir um resultado igual ao da Figura 6.1, que foi extraída da documentação, as cores de cada componente/região de enriquecimento são iguais.

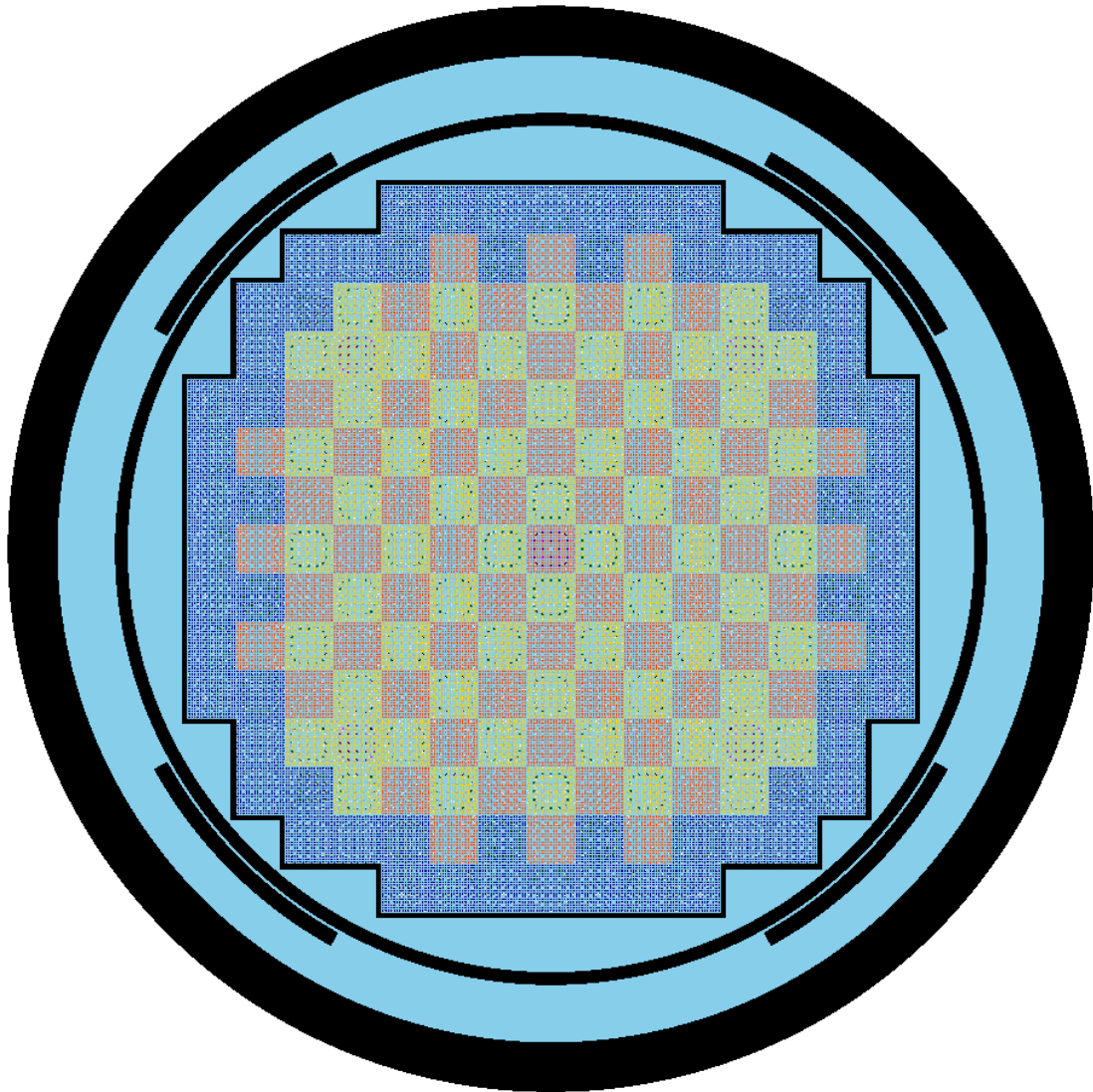


Figura 6.11: Núcleo do BEAVRS plotado com OpenMC: componentes estruturais e regiões de enriquecimento.

Na ampliação da Figura 6.11 os elementos que contém do Sistema de Controle D estão destacados. E mudando a cor da região de maior enriquecimento (3.1%), de azul escuro para uma cor mais clara, é possível visualizar os elementos com 6 e 15 pinos de veneno queimável distribuídos na periferia do núcleo.

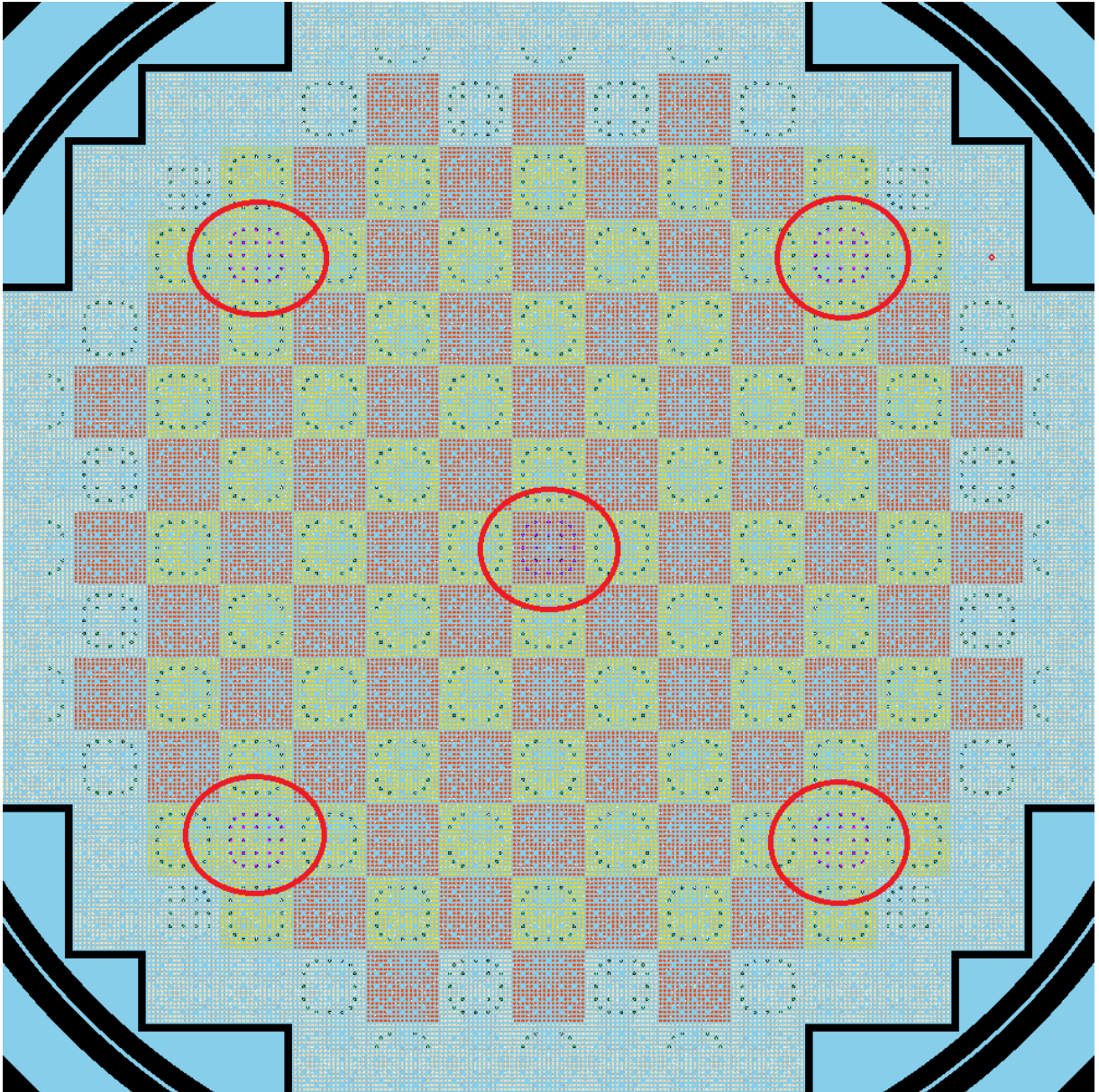


Figura 6.12: Ampliação da Figura 6.11; em destaque, os elementos que contém o Sistema de controle D, o único acionado na simulação do benchmark.

7 Simulação

As simulações dos núcleos do reator proposto (Figura 5.15) e do benchmark (Figura 6.11) serão detalhadas a seguir, mostrando como ocorre e como os inputs necessários aparecem durante a simulação. Os arquivos de simulação inteiros do reator e do benchmark estão disponíveis no Anexo I. No capítulo seguinte, de análise de resultados, os desdobramentos da simulação serão discutidos.

7.1 Benchmark

Antes de iniciar a simulação foi necessário escolher a fonte de início. Conforme a Figura 5.4, esta fonte foi posicionada exatamente no meio do sistema, arbitrariamente definido como origem dos eixos coordenados.

A simulação do núcleo do benchmark da Figura 6.11 utilizou uma fonte inicial uniformemente distribuída, cujas dimensões abrangem toda a área fissil do núcleo. Escolhida a fonte inicial, a simulação se inicia acionando o método `openmc.run`. Os arquivos xml são lidos, e, depois, as seções de choque dos nuclídeos declarados no arquivo de materiais são listadas. Terminada esta etapa, começa a simulação já com a indicação do modo que foi setado: **Eigenvalue**. O arquivo de simulação começa a ser escrito e três colunas são imediatamente definidas: 1^a mostrando o ciclo/número de gerações por ciclo; 2^a mostrando o valor de k para cada ciclo; 3^a mostrando o valor médio de k e seu desvio padrão para cada ciclo.

O atributo `Settings.inactive` nesta simulação foi 10. Isso significa, que nos 10 primeiros ciclos, os valores médios de k (**Average k**) foram descartados, para depois, imediatamente no ciclo n^o11, começarem a ser plotados. O valor de k_{eff} começa muito baixo e vai convergindo à medida que a simulação evolui. Quando o número de ciclos setado é atingido, o OpenMC escreve o arquivo `statepoint.*.h5`, onde o asterisco indica o número setado de ciclos; este arquivo será utilizado no processamento dos resultados. O número de ciclos foi setado em 100 com o atributo `Settings.batches`. A simulação do

núcleo do benchmark foi setada com o atributo `Settings.particle`, primeiramente, em 1000, depois 10000, e finalmente, 100000 partículas/ciclo.

Ao final da simulação, são exibidas as estatísticas de tempo, detalhando o tempo necessário para cada etapa, e os resultados da média e desvio padrão de k_{eff} (usando três estimadores diferentes e o estimador combinado) e da fração de fuga de nêutrons do sistema (Leakage Fraction). Para fins práticos, apenas a parte dos resultados da simulação será apresentada, e, diferente do formato no arquivo de simulação, em forma de tabela.

RESULTADOS	
k-effective (Collision)	1.01905 +/- 0.00397
k-effective (Track-length)	1.01522 +/- 0.00492
k-effective (Absorption)	1.01067 +/- 0.00335
Combined k-effective	1.01414 +/- 0.00307
Leakage Fraction	0.00002 +/- 0.00002

Tabela 7.1: Simulação do benchmark com 1000 partículas/ciclo com fonte inicial em formato de caixa.

Aumentando-se o número de partículas/ciclo de 1000 para 10000, a variância diminuiu e o desvio padrão dos valores de k_{eff} , conseqüentemente, diminuiu. A seguir, os resultados da simulação com 10000 partículas/ciclo.

RESULTADOS	
k-effective (Collision)	1.00984 +/- 0.00118
k-effective (Track-length)	1.00874 +/- 0.00141
k-effective (Absorption)	1.01112 +/- 0.00110
Combined k-effective	1.01035 +/- 0.00094
Leakage Fraction	0.00004 +/- 0.00001

Tabela 7.2: Simulação do benchmark com 10000 partículas/ciclo com fonte inicial em formato de caixa.

Aumentando-se o número de partículas/ciclo de 10000 para 100000, a variância diminui ainda mais.

RESULTADOS	
k-effective (Collision)	1.00965 +/- 0.00039
k-effective (Track-length)	1.00954 +/- 0.00042
k-effective (Absorption)	1.00970 +/- 0.00037
Combined k-effective	1.00966 +/- 0.00032
Leakage Fraction	0.00003 +/- 0.00000

Tabela 7.3: Simulação do benchmark com 100000 partículas/ciclo com fonte inicial em formato de caixa.

A fuga de nêutrons (Leakage Fraction) do benchmark é praticamente nula devido ao enorme inventário água, o agente espalhador/refletor. O tamanho do benchmark resulta numa maior economia de nêutrons. Na seção seguinte, veremos como o tamanho do reator proposto influencia a fuga de nêutrons na simulação.

7.2 Reator Proposto

A simulação do núcleo do reator proposto baseado no CAREM 25 da Figura 5.15 ocorreu da mesma maneira como a do núcleo do benchmark na seção anterior: atributo `Settings.inactive` setado em 10; atributo `Settings.batches` setado em 100; e o atributo `Settings.particle` setado, primeiramente, em 1000, depois 10000, e finalmente, 100000 partículas/ciclo.

RESULTADOS	
k-effective (Collision)	1.05949 +/- 0.00327
k-effective (Track-length)	1.05887 +/- 0.00373
k-effective (Absorption)	1.06639 +/- 0.00349
Combined k-effective	1.06231 +/- 0.00274
Leakage Fraction	0.03532 +/- 0.00086

Tabela 7.4: Simulação do reator proposto com 1000 partículas/ciclo com fonte inicial em formato de caixa.

RESULTADOS	
k-effective (Collision)	1.06376 +/- 0.00134
k-effective (Track-length)	1.06277 +/- 0.00151
k-effective (Absorption)	1.06248 +/- 0.00138
Combined k-effective	1.06287 +/- 0.00121
Leakage Fraction	0.03595 +/- 0.00023

Tabela 7.5: Simulação do reator proposto com 10000 partículas/ciclo com fonte inicial em formato de caixa.

RESULTADOS	
k-effective (Collision)	1.06290 +/- 0.00101
k-effective (Track-length)	1.06312 +/- 0.00103
k-effective (Absorption)	1.06294 +/- 0.00097
Combined k-effective	1.06297 +/- 0.00098
Leakage Fraction	0.03587 +/- 0.00050

Tabela 7.6: Simulação do reator proposto com 100000 partículas/ciclo com fonte inicial em formato de caixa.

Como esperado, a fuga de nêutrons do reator proposto foi maior comparada à do benchmark. O reator proposto é baseado num SMR, e, portanto, também se trata de

um SMR. Por isso, seu menor inventário água, o agente espalhador/refletor, resulta numa menor economia de nêutrons.

Como forma de avaliar a influência da fonte inicial, para o reator proposto, também foram feitas simulações utilizando uma fonte inicial pontual posicionada no meio do reator. Com os mesmos parâmetros de simulação da fonte inicial uniformemente distribuída: atributo `Settings.inactive` setado em 10; atributo `Settings.batches` setado em 100; e o atributo `Settings.particle` setado, primeiramente, em 1000, depois 10000, e finalmente, 100000 partículas/ciclo.

RESULTADOS	
k-effective (Collision)	1.05799 +/- 0.00382
k-effective (Track-length)	1.05767 +/- 0.00481
k-effective (Absorption)	1.06190 +/- 0.00367
Combined k-effective	1.06009 +/- 0.00329
Leakage Fraction	0.03348 +/- 0.00091

Tabela 7.7: Simulação do reator proposto com 1000 proposto partículas/ciclo com fonte inicial pontual.

RESULTADOS	
k-effective (Collision)	1.06285 +/- 0.00151
k-effective (Track-length)	1.06262 +/- 0.00165
k-effective (Absorption)	1.06027 +/- 0.00141
Combined k-effective	1.06120 +/- 0.00134
Leakage Fraction	0.03360 +/- 0.00057

Tabela 7.8: Simulação do reator proposto com 10000 partículas/ciclo com fonte inicial pontual.

RESULTADOS	
k-effective (Collision)	1.06303 +/- 0.00097
k-effective (Track-length)	1.06365 +/- 0.00104
k-effective (Absorption)	1.06308 +/- 0.00087
Combined k-effective	1.06301 +/- 0.00089
Leakage Fraction	0.03395 +/- 0.00052

Tabela 7.9: Simulação do reator proposto com 100000 partículas/ciclo com fonte inicial pontual.

A análise dos arquivos de simulação permite observar que a convergência do fator de multiplicação é mais estável para a fonte uniformemente distribuída que para a pontual, implicando num menor tempo de simulação.

8 Resultados

8.1 Convergência do fator de multiplicação efetivo

Todos os resultados dos cálculos Monte Carlo são estimativas da média de alguma variável randômica que queremos avaliar. Como primeira discussão, é necessário interpretar o processo estocástico das contagens à luz das incertezas associadas a ele. O primeiro ponto importante diz respeito à quantidade de partículas simuladas por ciclo. No gráfico da Figura 8.1, para a simulação do reator, a convergência do fator de multiplicação efetivo é mais estável quanto maior o número de partículas simuladas. Esta observação estatística diz que valores médios esperados de processos estocásticos que são realizados inúmeras vezes convergem para um certo valor médio. Se considerarmos cada ciclo independente do outro, e k a variável aleatória de média x , a média amostral converge para a média verdadeira: é a Lei dos Grandes Números, definida pela Equação 8.1.

$$\lim_{n \rightarrow \infty} P(|\bar{k}_n - x| > \epsilon) = 0, \epsilon > 0. \quad (8.1)$$

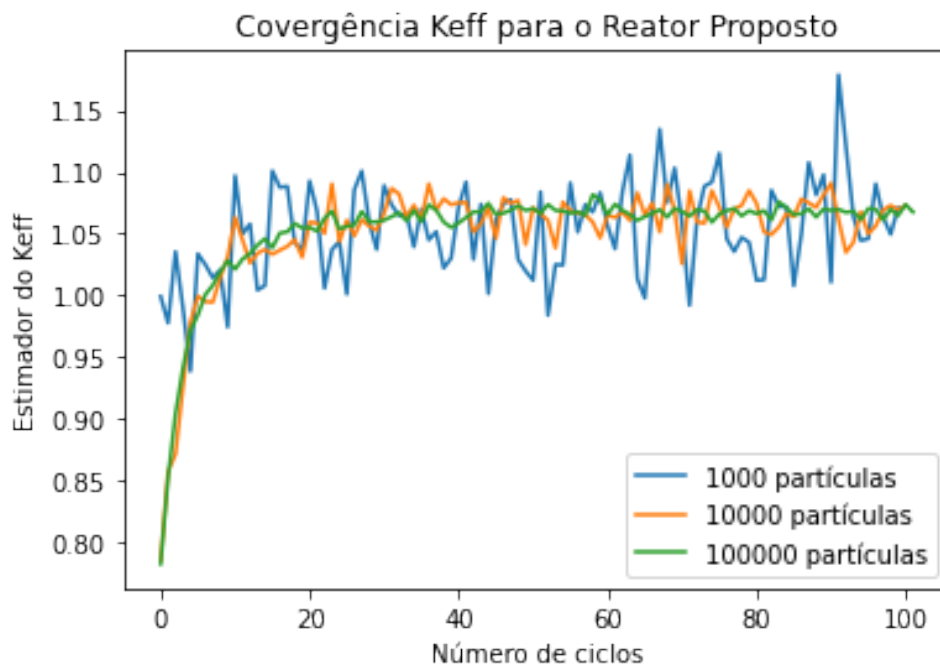


Figura 8.1: Gráfico de convergência do k_{eff} plotado usando OpenMC.

Pelo gráfico, na simulação com 100000 partículas por ciclo, a convergência do k_{eff} é estável que nas simulações com 10000 e 1000 partículas por ciclo, respectivamente, sem mencionar a menor flutuação estatística do k_{eff} à medida que o número de partículas por ciclo aumenta. Em concordância com os valores de k_{eff} obtidos no capítulo anterior, observamos a diminuição do desvio padrão para todos os estimadores, como mostra a Tabela 8.1.

partículas/ciclo	1000	10000	100000
k-effective (Collision)	1.05988+/-0.00429	1.06156+/-0.00150	1.06290+/-0.00101
k-effective (Track-length)	1.05935+/-0.00473	1.06222+/-0.00167	1.06312+/-0.00103
k-effective (Absorption)	1.06452+/-0.00381	1.06396+/-0.00134	1.06294+/-0.00097
Combined k-effective	1.06253+/-0.00343	1.06337+/-0.00130	1.06297+/-0.00098
Leakage Fraction	0.03471+/-0.00070	0.03332+/-0.00051	0.03387+/-0.00050

Tabela 8.1: Valores de k_{eff} vs número de partículas/ciclo - reator proposto.

O Teorema Central do Limite (TLC), explica a diminuição da variância à medida que o número de partículas/ciclo aumenta. Na verdade, é a média amostral que converge para uma distribuição normal à medida que a população do experimento aumenta. Novamente, considerando um ciclo independente do outro e k_{eff} como a variável aleatória com média x e variância $\text{Var}(k_i)=\sigma^2<\infty$, a variável aleatória converge para uma distribuição normal e padronizada, conforme a Equação 8.2.

$$\sqrt{n} \left(\frac{1}{n} \sum_{i=1}^n \bar{k}_n - x \right) \xrightarrow{d} N(0, \sigma^2). \quad (8.2)$$

A variância é uma medida do quão próximo estamos do valor da contagem; trata-se de uma medida de dispersão. Isso implica que simulações com variâncias menores conduzem a valores de k_{eff} mais precisos. Uma relação mais didática para a variância e o número de partículas/ciclo é definida pela Equação 8.3.

$$\text{Var}(k_i) \propto \frac{1}{\sqrt{n}}, n = \text{partículas/ciclo}. \quad (8.3)$$

A gráfico da Figura 8.2 mostra este comportamento para as simulações do benchmark à medida que o número de partículas/ciclo aumenta de 1000, para 10000, e depois, para 100000. Da mesma forma que para o reator proposto, a linha verde, correspondente à simulação com 100000 partículas/ciclo, comparando com as outras simulações, apresenta uma flutuação estatística bem menor. E em concordância com os valores de k_{eff} obtidos no capítulo anterior, observamos a diminuição do desvio padrão para todos os estimadores, como mostra a Tabela 8.2.

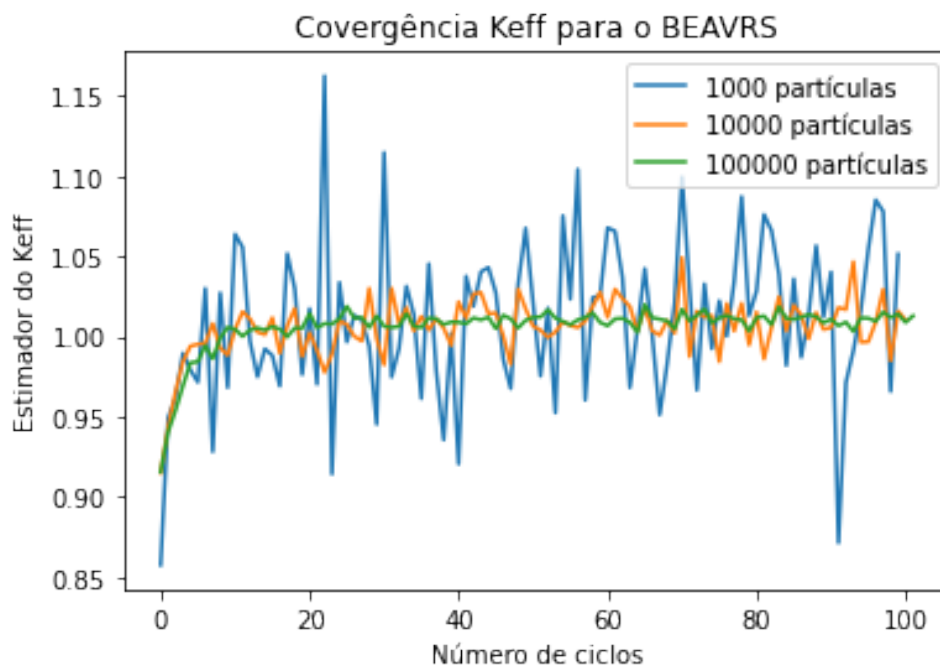


Figura 8.2: Gráfico de Convergência do k_{eff} plotado usando OpenMC.

partículas/ciclo	1000	10000	100000
k-effective (Collision)	1.01905+/-0.00397	1.00984+/-0.00118	1.00965+/-0.00039
k-effective (Track-length)	1.01522+/-0.00492	1.00874+/-0.00141	1.00954+/-0.00042
k-effective (Absorption)	1.01067+/-0.00335	1.01112+/-0.00110	1.00970+/-0.00037
Combined k-effective	1.01414+/-0.00307	1.01035+/-0.00094	1.00966+/-0.00032
Leakage Fraction	0.00002+/-0.00002	0.00004+/-0.00001	0.00003+/-0.00000

Tabela 8.2: Valores de k_{eff} vs número de partículas/ciclo - benchmark.

Quanto à influência da fonte inicial de simulação escolhida, o gráfico da Figura 8.3 mostra a convergência do k_{eff} comparando as fontes pontual e uniformemente distribuída (Box) para a simulação do reator:

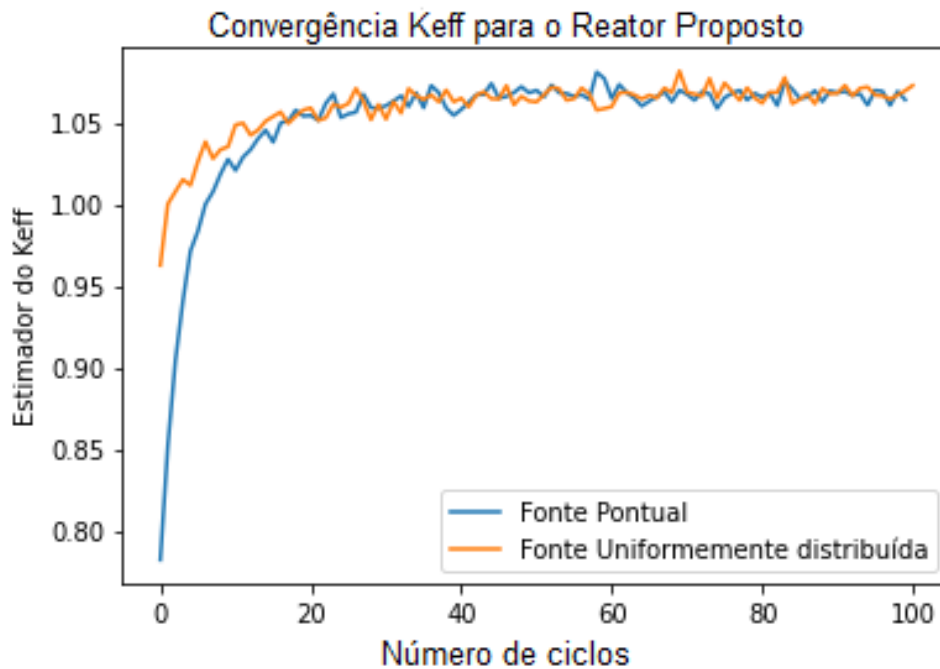


Figura 8.3: Comparação entre as fontes pontual e uniformemente distribuída plotada usando OpenMC.

A fonte uniformemente distribuída já começa de um valor mais próximo para qual vai convergir do que a fonte pontual.

8.2 Fórmula dos 6 fatores para k_{eff}

8.2.1 Benchmark

O valor do k_{eff} do benchmark (média e desvio padrão) pode ser obtido ao final da simulação, como já mostrado, ou através de contagens derivadas. Ao término da simulação o OpenMC escreve o arquivo `statepoint.*.h5`, que é um arquivo binário como todas as contagens setadas, próprio para ser utilizado em vetores. Alocando-o num objeto de `openmc.StatePoint` e com o método `get_tally` essas contagens podem ser coletadas e guardadas em variáveis. Através das operações de soma, subtração, divisão e multiplicação entre as contagens derivadas determina-se o valor de k_{eff} . É desta maneira que se obtém o k_{eff} pela Fórmula dos 6 fatores, conforme a Figura 8.4, que mostra uma célula do Jupyter onde o arquivo é carregado.

```
In [27]: # Load the statepoint file
sp = openmc.StatePoint('statepoint.100.h5')
```

```
In [28]: # Get the fission and absorption rate tallies
fiss_rate = sp.get_tally(name='fiss. rate')
abs_rate = sp.get_tally(name='abs. rate')

# Get the leakage tally
leak = sp.get_tally(name='leakage')
leak = leak.summation(filter_type=openmc.MeshSurfaceFilter, remove_filter=True)

# Compute k-infinity using tally arithmetic
keff = fiss_rate / (abs_rate + leak)
keff.get_pandas_dataframe()
```

```
Out[28]:
```

	nuclide	score	mean	std. dev.
0	total (nu-fission / (absorption + current))		0.983288	0.001946

Figura 8.4: k_{eff} obtido pela Equação 8.4 para o benchmark.

O parâmetro k_{eff} é conhecido na Física de reatores pela Equação 8.4, e implementado no OpenMC como mostrado na Figura 8.4.

$$k_{\text{eff}} = \frac{\langle \nu \Sigma_f \phi \rangle}{\langle \Sigma_a \phi \rangle + \langle L \rangle}. \quad (8.4)$$

O k_{eff} em termos dos 6 fatores é obtido pela Equação 8.5, onde: \mathbf{p} é a probabilidade de escape ressonante (no OpenMC será contada na faixa térmica); ϵ é o fator de fissão rápida; \mathbf{f} é a utilização térmica do combustível; η é o número de nêutrons de fissão produzidos por absorção no combustível; \mathbf{P}_{FNL} é a probabilidade de não fuga rápida; e \mathbf{P}_{TNL} é a probabilidade de não fuga térmica.

$$k_{\text{eff}} = p\epsilon f\eta P_{\text{FNL}}P_{\text{TNL}}. \quad (8.5)$$

A seguir, o computo de cada um dos 6 fatores que serão utilizados para o cálculo de k_{eff} do benchmark via Equação 8.5 usando as contagens do OpenMC.

1º: \mathbf{p} , probabilidade de escape ressonante, definida pela Equação 8.6, e implementada pelo OpenMC como mostrado na Figura 8.5.

$$p = \frac{\langle \Sigma_a \phi \rangle_T + \langle L \rangle_T}{\langle \Sigma_a \phi \rangle + \langle L \rangle_T}, T : \textit{termico}. \quad (8.6)$$

```
In [29]: # Compute resonance escape probability using tally arithmetic
therm_abs_rate = sp.get_tally(name='therm. abs. rate')
thermal_leak = sp.get_tally(name='thermal leakage')
thermal_leak = thermal_leak.summation(filter_type=openmc.MeshSurfaceFilter, remove_filter=True)
res_esc = (therm_abs_rate + thermal_leak) / (abs_rate + thermal_leak)
res_esc.get_pandas_dataframe()
```

```
Out[29]:
```

	energy low [eV]	energy high [eV]	nuclide	score	mean	std. dev.
0	0.0	0.625	total	((absorption + current) / (absorption + current))	0.69372	0.00124

Figura 8.5: Probabilidade de escape ressonante - bechmark.

2º: ϵ , fator de fissão rápida, definido pela Equação 8.7, e implementado pelo OpenMC como mostrado na Figura 8.6.

$$\epsilon = \frac{\langle \nu \Sigma_f \phi \rangle}{\langle \nu \Sigma_f \phi \rangle_T}. \quad (8.7)$$

```
In [30]: # Compute fast fission factor using tally arithmetic
therm_fiss_rate = sp.get_tally(name='therm. fiss. rate')
fast_fiss = fiss_rate / therm_fiss_rate
fast_fiss.get_pandas_dataframe()
```

Out[30]:

	energy low [eV]	energy high [eV]	nuclide	score	mean	std. dev.
0	0.0	0.625	total (nu-fission / nu-fission)	1.223142	0.002645	

Figura 8.6: Fator de fissão rápida - benchmark.

3º: f , utilização térmica do combustível definida pela Equação 8.8, e implementada pelo OpenMC como mostrado na Figura 8.7.

$$f = \frac{\langle \Sigma_a \phi \rangle_T^F}{\langle \Sigma_a \phi \rangle_T}, F : fuel(combustivel). \quad (8.8)$$

```
In [31]: # Compute thermal flux utilization factor using tally arithmetic
fuel_therm_abs_rate = sp.get_tally(name='fuel therm. abs. rate')
therm_util = fuel_therm_abs_rate / therm_abs_rate
therm_util.get_pandas_dataframe()
```

Out[31]:

	energy low [eV]	energy high [eV]	material	nuclide	score	mean	std. dev.
0	0.0	0.625	15	total (absorption / absorption)	0.238926	0.000544	

Figura 8.7: Utilização Térmica do Combustível - benchmark.

4º: η , número de nêutrons de fissão produzidos por absorção no combustível, definido pela Equação 8.9, e implementado pelo OpenMC como mostrado na Figura 8.8.

$$\eta = \frac{\langle \nu \Sigma_f \phi \rangle_T}{\langle \Sigma_a \phi \rangle_T^F}. \quad (8.9)$$

```
In [32]: # Compute neutrons produced per absorption (eta) using tally arithmetic
eta = therm_fiss_rate / fuel_therm_abs_rate
eta.get_pandas_dataframe()
```

Out[32]:

	energy low [eV]	energy high [eV]	material	nuclide	score	mean	std. dev.
0	0.0	0.625	15	total (nu-fission / absorption)	4.976627	0.012068	

Figura 8.8: Número de nêutrons de fissão produzidos por absorção no combustível - benchmark.

5º: P_{FNL} , probabilidade de não fuga rápida, definida pela Equação 8.10, e implementada pelo OpenMC como mostrado na Figura 8.9.

$$P_{\text{FNL}} = \frac{\langle \Sigma_a \phi \rangle + \langle L \rangle_T}{\langle \Sigma_a \phi \rangle + \langle L \rangle}. \quad (8.10)$$

```
In [33]: p_fnl = (abs_rate + thermal_leak) / (abs_rate + leak)
p_fnl.get_pandas_dataframe()
```

Out[33]:

	energy low [eV]	energy high [eV]	nuclide	score	mean	std. dev.
0	0.0	0.625	total ((absorption + current) / (absorption + current))	0.980627	0.001724	

Figura 8.9: probabilidade de não fuga rápida - benchmark.

6º: P_{TNL} , probabilidade de não fuga térmica 6, definida pela Equação 8.11, e implementada pelo OpenMC como mostrado na Figura 8.10.

$$P_{\text{TNL}} = \frac{\langle \Sigma_a \phi \rangle_T}{\langle \Sigma_a \phi \rangle_T + \langle L \rangle_T}. \quad (8.11)$$

```
In [34]: p_tnl = therm_abs_rate / (therm_abs_rate + thermal_leak)
p_tnl.get_pandas_dataframe()
```

```
Out[34]:
```

	energy low [eV]	energy high [eV]	nuclide	score	mean	std. dev.
0	0.0	0.625	total (absorption / (absorption + current))	0.993843	0.001993	

Figura 8.10: Probabilidade de não fuga térmica - benchmark.

Finalmente, o cálculo do k_{eff} calculado com os valores dos 6 fatores obtidos separadamente é apresentado na Figura 8.11.

```
In [35]: keff = res_esc * fast_fiss * therm_util * eta * p_fnl * p_tnl
keff.get_pandas_dataframe()
```

```
Out[35]:
```

	energy low [eV]	energy high [eV]	material	nuclide	score	mean	std. dev.
0	0.0	0.625	15	total ((((((absorption + current) / (absorption + c...	0.983288	0.005018	

Figura 8.11: k_{eff} pela fórmula dos 6 fatores - benchmark.

A comparação do valor apresentado no final da simulação (Combined k-effective) com os valores calculados pelas contagens derivadas, é apresentada na Tabela 8.3 para o benchmark.

	Combined k-effective
Simulação	1.01032 +/- 0.00095
Fórmula Eq. 8.4	0.98329 +/- 0.00195
Fórmula 6 fatores	0.98329 +/- 0.00502

Tabela 8.3: Comparação entre simulação e contagens derivadas para o k_{eff} do benchmark.

O desvio padrão da simulação é comparável ao do k_{eff} obtido a partir da Equação 8.4. Nesta maneira de cálculo do k_{eff} as incertezas associadas a cada contagem são propagadas para o resultado final devido às operações de soma e divisão. O valor da média, difere na terceira casa depois da vírgula pois o valor da simulação é decorrente de um processo de convergência de k , ao contrário do resultado da Equação 8.4, o que indica

valores mais precisos de k são obtidos via simulação direta. O mesmo raciocínio vale para a média da Fórmula dos 6 Fatores, que também difere da simulação, porém com desvio padrão maior até mesmo do valor obtido pela Equação 8.4.

8.2.2 Reator Proposto

O parâmetro k_{eff} do reator proposto também pode ser obtido pela Equação 8.4, e é apresentado na Figura 8.12.

```
In [124]: # Get the fission and absorption rate tallies
fiss_rate = sp.get_tally(name='fiss. rate')
abs_rate = sp.get_tally(name='abs. rate')

# Get the leakage tally
leak = sp.get_tally(name='leakage')
leak = leak.summation(filter_type=openmc.MeshSurfaceFilter,
                      remove_filter=True)

# Compute k-infinity using tally arithmetic
keff = fiss_rate / (abs_rate + leak)
keff.get_pandas_dataframe()
```

Out[124]:

	nuclide	score	mean	std. dev.
0	total (nu-fission / (absorption + current))		1.065645	0.001949

Figura 8.12: k_{eff} obtido pela Equação 8.4 para o reator proposto.

A seguir, o computo de cada um dos 6 fatores que serão utilizados para o cálculo de k_{eff} do reator proposto via Equação 8.5 usando as contagens do OpenMC.

1º: p , probabilidade de escape ressonante, definida pela Equação 8.6, e implementada pelo OpenMC como mostrado na Figura 8.13.

```
In [125]: # Compute resonance escape probability using tally arithmetic
therm_abs_rate = sp.get_tally(name='therm. abs. rate')
thermal_leak = sp.get_tally(name='thermal leakage')
thermal_leak = thermal_leak.summation(filter_type=openmc.MeshSurfaceFilter,
                                     remove_filter=True)

res_esc = (therm_abs_rate + thermal_leak) / (abs_rate + thermal_leak)
res_esc.get_pandas_dataframe()
```

Out[125]:

	energy low [eV]	energy high [eV]	nuclide	score	mean	std. dev.
0	0.0	0.625	total	((absorption + current) / (absorption + current))	0.799643	0.001406

Figura 8.13: Probabilidade de escape ressonante - reator proposto.

2º: ϵ , fator de fissão rápida, definido, pela Equação 8.7, e implementado pelo OpenMC como mostrado na Figura 8.14.

```
In [126]: # Compute fast fission factor factor using tally arithmetic
therm_fiss_rate = sp.get_tally(name='therm. fiss. rate')
fast_fiss = fiss_rate / therm_fiss_rate
fast_fiss.get_pandas_dataframe()
```

Out[126]:

	energy low [eV]	energy high [eV]	nuclide	score	mean	std. dev.
0	0.0	0.625	total	(nu-fission / nu-fission)	1.131963	0.002416

Figura 8.14: Fator de fissão rápida - reator proposto.

3º: f , utilização térmica do combustível, definida pela Equação 8.8, e implementada pelo OpenMC como mostrado na Figura 8.15.

```
In [127]: # Compute thermal flux utilization factor using tally arithmetic
fuel_therm_abs_rate = sp.get_tally(name='fuel therm. abs. rate')
therm_util = fuel_therm_abs_rate / therm_abs_rate
therm_util.get_pandas_dataframe()
```

Out[127]:

	energy low [eV]	energy high [eV]	material	nuclide	score	mean	std. dev.
0	0.0	0.625	5	total	(absorption / absorption)	0.197299	0.000525

Figura 8.15: Utilização Térmica do Combustível - reator proposto.

4º: η , número de nêutrons de fissão produzidos por absorção no combustível, definido pela Equação 8.9, e implementado pelo OpenMC como mostrado na Figura 8.16.

```
In [128]: # Compute neutrons produced per absorption (eta) using tally arithmetic
eta = therm_fiss_rate / fuel_therm_abs_rate
eta.get_pandas_dataframe()
```

Out[128]:

	energy low [eV]	energy high [eV]	material	nuclide	score	mean	std. dev.
0	0.0	0.625	5	total (nu-fission / absorption)	6.175212	0.017238	

Figura 8.16: Número de nêutrons de fissão produzidos por absorção no combustível - reator proposto.

5º: P_{FNL} , probabilidade de não fuga rápida, definida pela Equação 8.10, e implementada pelo OpenMC como mostrado na Figura 8.17.

```
In [129]: p_fnl = (abs_rate + thermal_leak) / (abs_rate + leak)
p_fnl.get_pandas_dataframe()
```

Out[129]:

	energy low [eV]	energy high [eV]	nuclide	score	mean	std. dev.
0	0.0	0.625	total ((absorption + current) / (absorption + current))	0.971244	0.001563	

Figura 8.17: probabilidade de não fuga rápida - reator proposto.

6º: P_{TNL} , probabilidade de não fuga térmica, definida pela Equação 8.11, e implementada pelo OpenMC como mostrado na Figura 8.18.

```
In [130]: p_tnl = therm_abs_rate / (therm_abs_rate + thermal_leak)
p_tnl.get_pandas_dataframe()
```

Out[130]:

	energy low [eV]	energy high [eV]	nuclide	score	mean	std. dev.
0	0.0	0.625	total (absorption / (absorption + current))	0.994899	0.001904	

Figura 8.18: Probabilidade de não fuga térmica - reator proposto.

Finalmente, o cálculo do k_{eff} calculado com os valores dos 6 fatores obtidos separadamente é apresentado na Figura 8.19.

```
In [131]: keff = res_esc * fast_fiss * therm_util * eta * p_fnl * p_tnl
keff.get_pandas_dataframe()

Out[131]:
```

	energy low [eV]	energy high [eV]	material	nuclide	score	mean	std. dev.
0	0.0	0.625	5	total	(((((((absorption + current) / (absorption + c...	1.065645	0.005717

Figura 8.19: k_{eff} pela fórmula dos 6 fatores - reator proposto.

A comparação do valor apresentado no final da simulação (Combined k-effective) com o calculado pelas contagens derivadas é apresentada na Tabela 8.4.

	Combined k-effective
Simulação	1.06297 +/- 0.00098
Fórmula Eq. 8.4	1.06683 +/- 0.00091
Fórmula 6 fatores	1.06565 +/- 0.00572

Tabela 8.4: Comparação entre simulação e contagens derivadas para o k_{eff} do reator proposto.

A comparação entre os três resultados segue a mesma lógica do benchmark: o valor via simulação possui menor desvio padrão comparável ao obtido pela Equação 8.4, contudo, o valor da média, novamente, difere na terceira casa pois o valor da simulação é decorrente de um processo de convergência, ao contrário do resultado da Equação 8.4. A Fórmula dos 6 Fatores, para o caso do reator proposto, também produziu um resultado com desvio padrão que os demais.

8.3 Contagens de Fissão e Fluxo

8.3.1 Benchmark

Quando as contagens são setadas é possível associar às elas filtros específicos e malhas apropriadas. A célula da Figura 8.20 mostra os filtros e a malha associada à contagem de taxa de fissão (**fission**). A malha bidimensional (mesh: objeto de

`openmc.RegularMesh`) com 100 divisões em cada direção. Esta malha é atribuído do filtro `openmc.MeshFilter`, que serve para setar as contagens nas células da malha. Além deste, é também setado um filtro de energia (objeto de `openmc.EnergyFilter`) cujo atributo é uma lista com os intervalos de energia dos grupos térmico e rápido, `[0.0-0.625eV]` e `[0.625eV-20.0MeV]`, respectivamente. Terminada a simulação é possível plotar imagens correspondendo à seção transversal do reator, mostrando as regiões de fissão térmica e rápida. Para isso, foi utilizado o método `tally.get_pandas_dataframe()`, que gera automaticamente, do arquivo statepoint, as contagens na forma de vetores, através dos quais as imagens são geradas.

```
In [28]: # Create mesh which will be used for tally
mesh = openmc.RegularMesh()
mesh.dimension = [100, 100]
mesh.lower_left = [-172.0, -172.0]
mesh.upper_right = [+172.0, +172.0]

# Create mesh filter for tally
mesh_filter = openmc.MeshFilter(mesh)

# Instantiate energy Filter
energy_filter = openmc.EnergyFilter([0.0, 0.625, 20.0e6])

# Instantiate the Tally
tally = openmc.Tally(name='mesh tally')
tally.filters = [mesh_filter, energy_filter]
tally.scores = ['fission', 'nu-fission', 'prompt-nu-fission']

tallies.append(tally)
```

Figura 8.20: Filtros de Energia e Malha para as contagens do benchmark.

Na Figura 8.21, as imagens das contagens de fissão (**fission**) do benchmark para os dois grupos de energia são apresentadas. O benchmark possui as regiões mais claras no centro do reator, escurecendo em direção à periferia, onde estão os elementos mais enriquecidos. As regiões com maiores contagens de fissão ficam mais distribuídas em torno do centro, de menor enriquecimento, coincidindo com a região de maior fluxo, como mostrado na imagem do fluxo médio, Figura 8.22a.

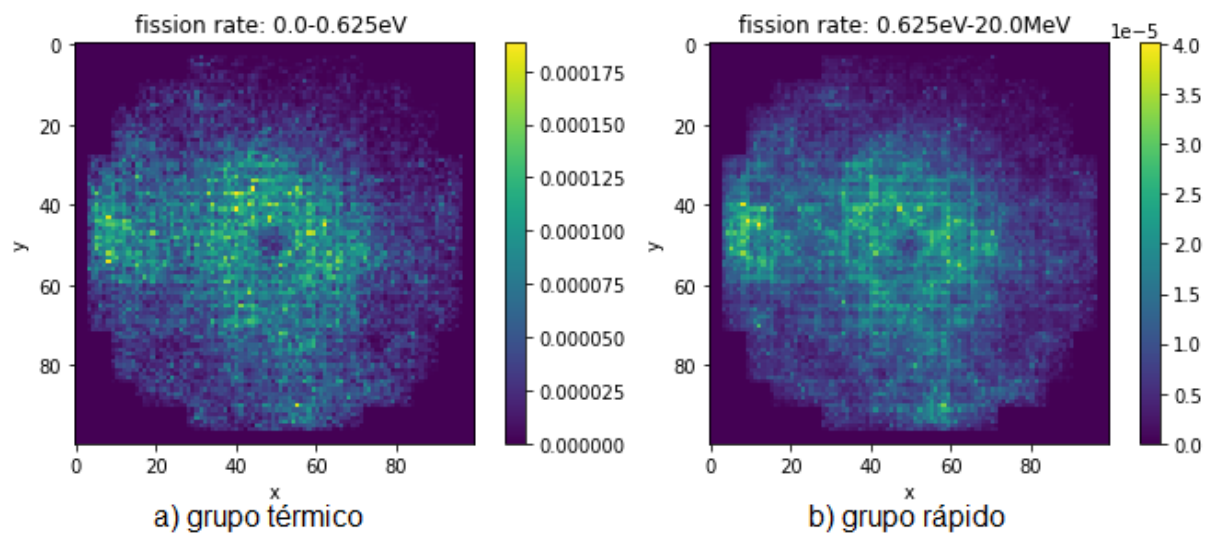


Figura 8.21: Contagens de fissão para dois grupos de energia - benchmark.

Nas Figuras 8.22a e 8.22b, a melhor definição permite visualizar 5 regiões escuras onde bancos de controle do Sistema D estão inseridos no núcleo do benchmark (conforme o mapa da Figura 6.3, do Sistema de Controle do BEAVRS).

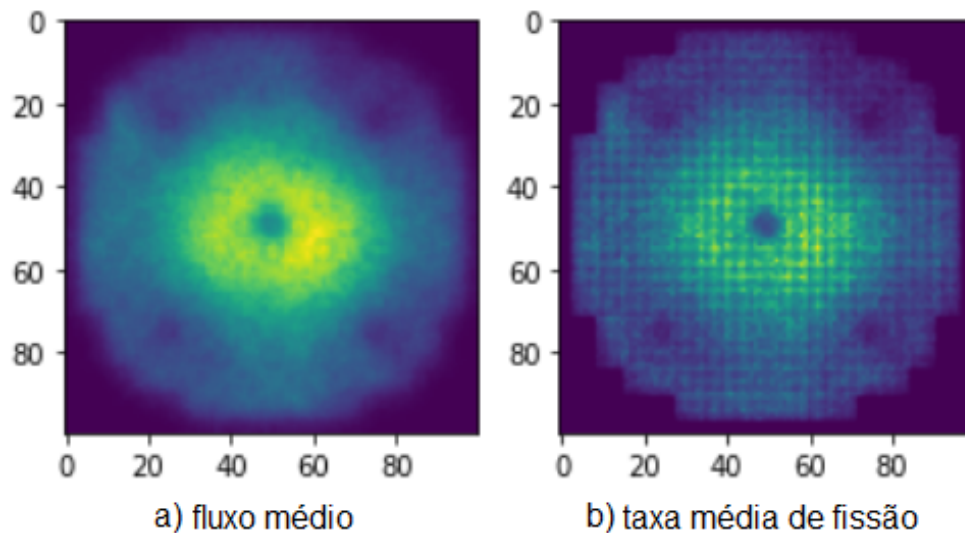


Figura 8.22: Fluxo médio do benchmark com malha 100 x 100.

A configuração dos elementos combustíveis do benchmark desempenha um papel na distribuição do fluxo. A configuração dos enriquecimentos do ciclo 1 no benchmark

(utilizada neste trabalho), é baseada num PWR/Westinghouse, e consiste em posicionar os elementos com menor enriquecimento na região de maior fluxo, queimando-os o máximo possível, e os elementos com maior enriquecimentos na região de menor fluxo, na periferia do núcleo. Durante a operação do reator, os elementos do centro passam por um intenso processo de queima, e os elementos da periferia, por serem expostos a um fluxo menor, queimam-se menos. Na recarga, os elementos da periferia e de suas proximidades são reposicionados mais ao centro e elementos frescos são posicionados na periferia.

A Figura 8.23, da documentação do BEAVRS, mostra como seria a distribuição de enriquecimento do ciclo 2 (Figura 8.23b), de recarga do benchmark, assim como a distribuição de varetas com veneno queimável nos ECs, que é diferente para o ciclo 2, como destacado nos números dos elementos na cor verde no ciclo 2.

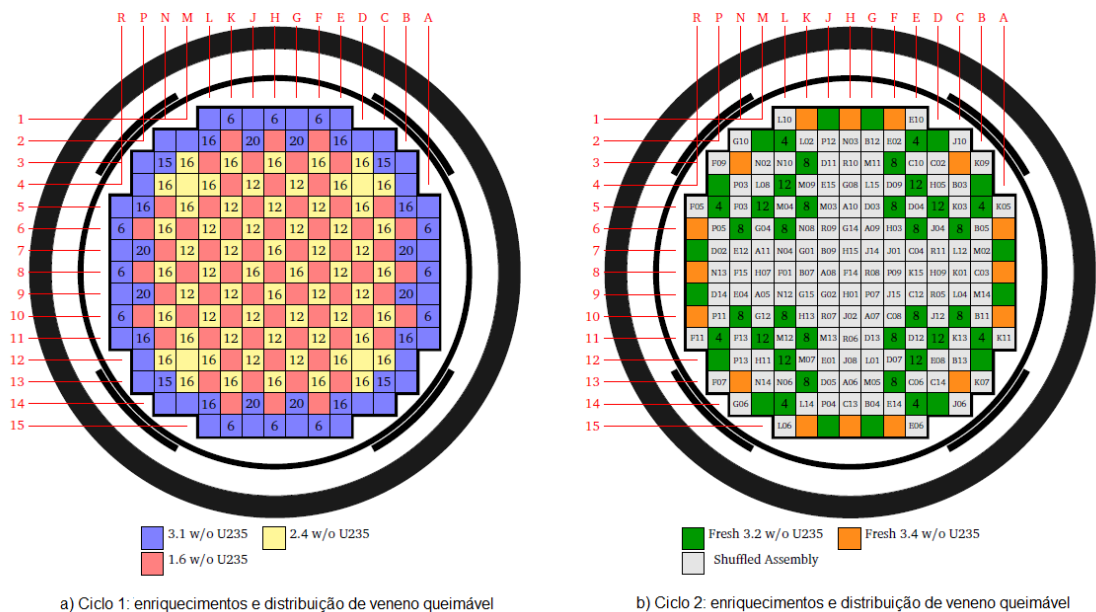


Figura 8.23: Ciclos de carregamento 1 e 2 do benchmark (HORELIK et al., 2020).

A Figura 8.24 apresenta como estão distribuídas as regiões de maior liberação de energia no núcleo do benchmark obtidas com as contagens normalizadas de fissão. A região de maior liberação de energia está localizada no centro do núcleo do benchmark, na região de fluxo mais intenso, e, por consequência, de queima máxima, circundada por

regiões onde o fluxo diminui radialmente até a periferia do núcleo, de menor queima. Como a liberação de energia está relacionado à potência, o benchmark tem uma distribuição de potência racional do ponto de vista do aproveitamento dos combustíveis devido ao arranjo dos enriquecimentos.

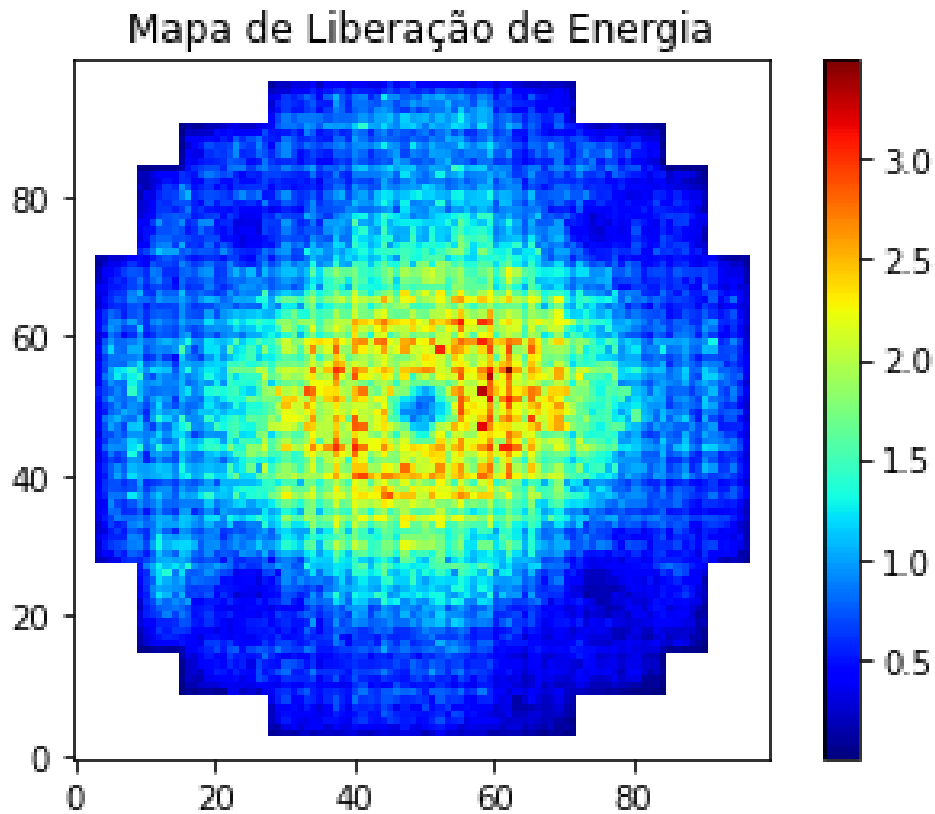


Figura 8.24: Mapa de liberação de energia no núcleo do benchmark.

8.3.2 Reator Proposto

A célula da Figura 8.25 mostra os filtros e a malha associada à contagem de taxa de fissão (**fission**) do reator proposto, utilizando a mesma malha bidimensional com 100 divisões em cada direção e os mesmos filtros que para o benchmark.


```

In [112]: # Create mesh which will be used for tally
mesh = openmc.RegularMesh()
mesh.dimension = [100, 100]
mesh.lower_left = [-75.0, -75.0]
mesh.upper_right = [+75.0, +75.0]

# Create mesh filter for tally
mesh_filter = openmc.MeshFilter(mesh)

# Instantiate energy Filter
energy_filter = openmc.EnergyFilter([0.0, 0.625, 20.0e6])

# Instantiate the Tally
tally = openmc.Tally(name='mesh tally')
tally.filters = [mesh_filter, energy_filter]
tally.scores = ['fission', 'nu-fission', 'prompt-nu-fission']

tallies.append(tally)

```

Figura 8.25: Filtros de Energia e Malha para as contagens do reator proposto.

Na Figura 8.26, as imagens das contagens de fissão (**fission**) para os dois grupos de energia são apresentadas. A escala de cor define as regiões escuras como não produtoras de nêutrons (água e absorvedores), e as regiões coloridas como produtoras de nêutrons (zonas de combustível). As zonas mais claras estão localizadas nas regiões mais enriquecidas e sem gadolínio e são simetricamente dispostas nas duas imagens. As escalas de cores das imagens indicam que as contagens das taxas de fissão do grupo rápido são menores que as taxas de fissão do grupo térmico.

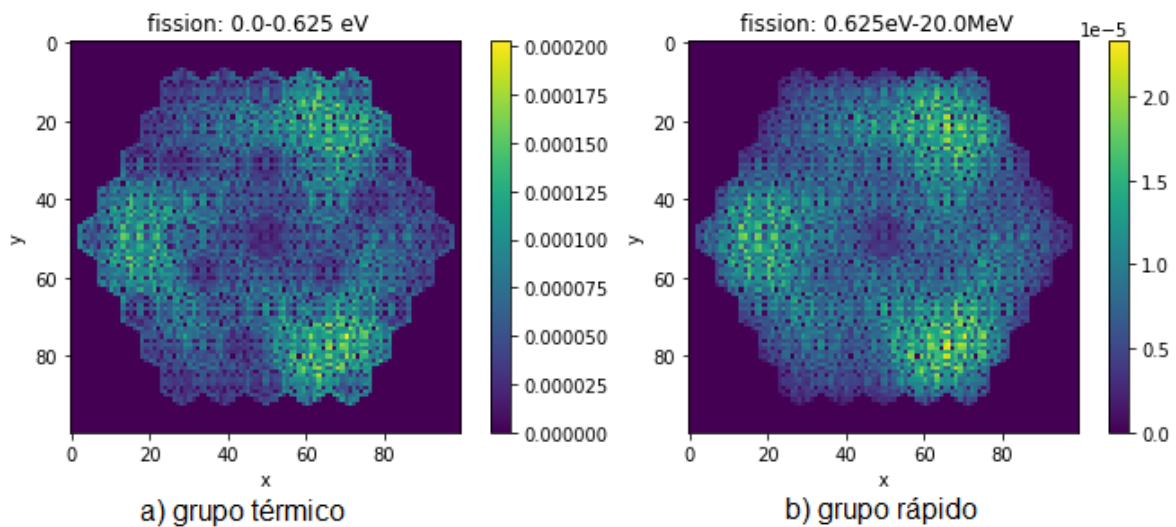


Figura 8.26: Contagens das taxa de fissão para dois grupos de energia - reator proposto.

Uma imagem do fluxo médio e da taxa média de fissão de uma seção transversal é gerada com a média e o desvio padrão do fluxo e da taxa de fissão ao longo das 100 células em cada direção. A Figura 8.27a mostra o fluxo médio na região dentro do vaso de pressão, e o padrão de cores segue o da figura anterior: as regiões mais escuras não há fluxo ou é muito baixo, e as mais claras, onde as contagens são mais altas; esta maneira de plotagem, no entanto, não contempla o atributo de escala de cores, apesar do padrão ser o mesmo. A maior da taxa média de fissão, Figura 8.27b, coincide justamente com as regiões de maior contagem de fissão, como visto nas Figuras 8.26a e 8.26b.

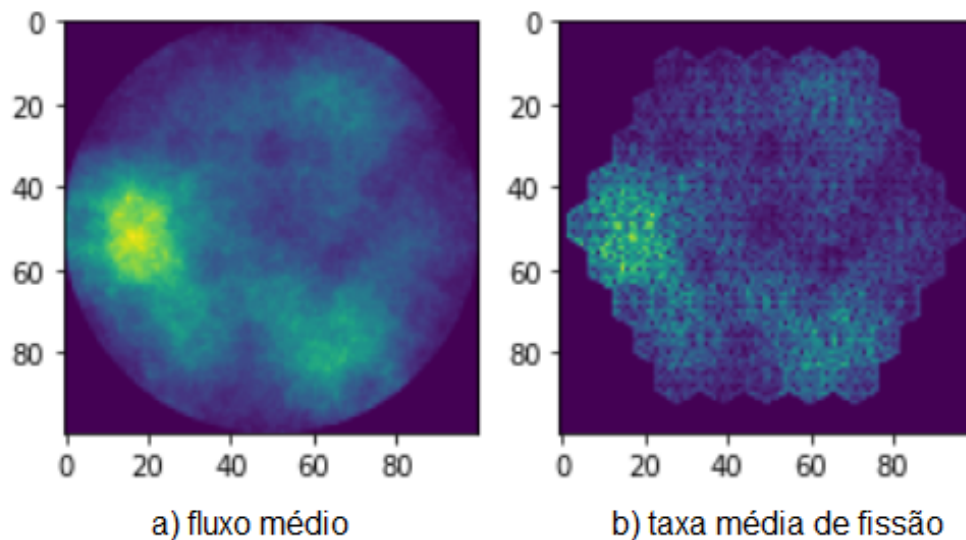


Figura 8.27: Fluxo médio e taxa média de fissão do reator proposto.

A configuração do reator dá origem a três áreas de fluxo intenso dispostas simetricamente, coincidindo com as regiões de maior enriquecimento e sem presença de gadolínio. A alta seção de choque de captura do gadolínio é responsável pela baixa de contagem de fissão nos elementos combustíveis onde ele é utilizado. Na Figura 8.26a, as regiões onde as contagens de fissão do grupo térmico são baixas estão localizadas em elementos combustíveis com gadolínio que possuem altas seções de choque de captura de nêutrons térmicos, mais precisamente os isótopos Gd^{155} e Gd^{157} , como observou TASHAKOR et al. (2017). Por isso, há uma competição entre os isótopos Gd^{155} e Gd^{157} e o

isótopo U^{235} pelos dos nêutrons térmicos.

As seções de choque dos núclídeos podem ser plotados usando o OpenMC através da subclasse `openmc.data.IncidentNeutron`. Depois de importar os dados nucleares (com extensão `.h5`) as seções de choque são plotadas em gráficos log-log. A Figura mostram as seções de choque de captura radiativa dos isótopos Gd^{155} e Gd^{157} , respectivamente, e as seções de choque de fissão dos isótopos U^{235} e U^{238} , respectivamente.

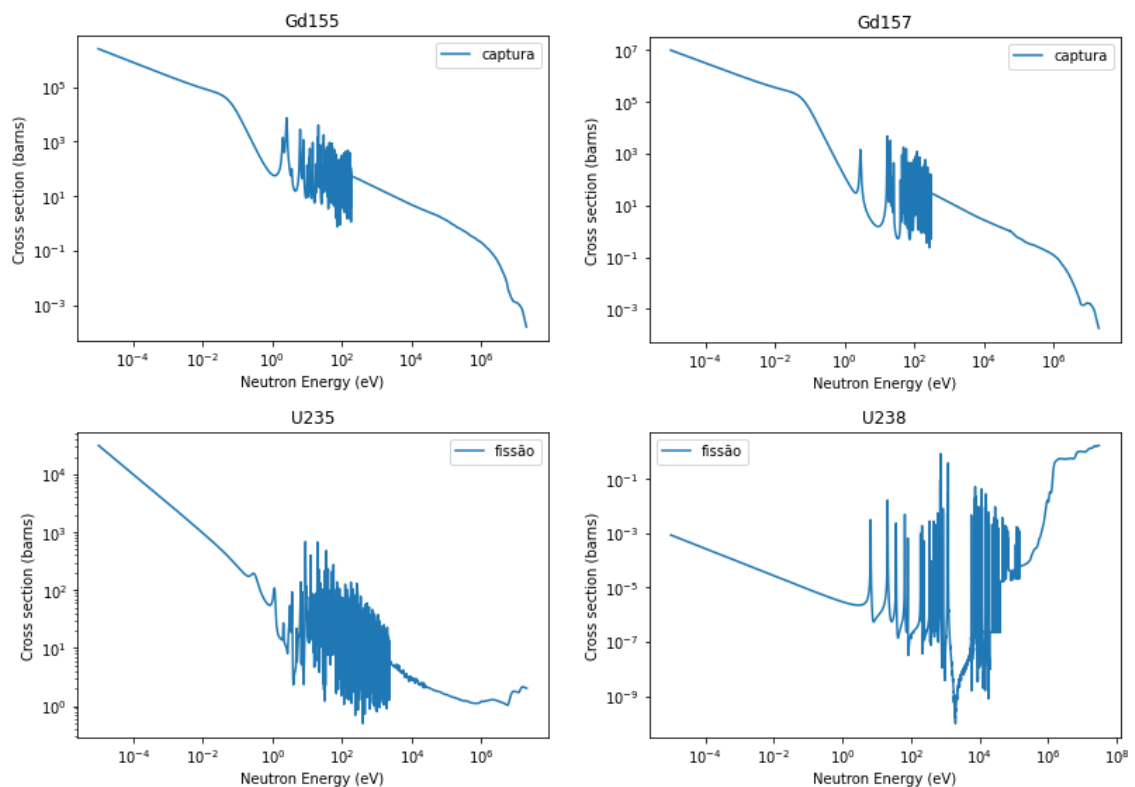


Figura 8.28: Seções de choque de captura dos isótopos Gd^{155} e Gd^{157} e de fissão dos isótopos U^{235} e U^{238} .

A análise da seção de choque de fissão do U^{235} mostra que ela é maior para a faixa de energia do nêutron para grupo térmico, setada pelo filtro de energia entre 0.0eV e 0.625eV em relação ao U^{238} ; o U^{238} , por sua vez, possui seção de choque de fissão menor para esta mesma faixa de energia do nêutron. Como o conceito de seção de choque é probabilístico, a maior probabilidade de ocorrência de fissão para o grupo térmico é associada ao U^{235} . As seções de choque de captura radiativa do Gd^{155} e Gd^{157} são maiores para a faixa de energia do grupo térmico, o que explica as regiões de baixa contagem de

fissão das Figuras 8.26a e 8.26b

Com as contagens de fissão e fluxo concluídas, a próxima etapa é determinar como as regiões de maior liberação de energia estão distribuídas no núcleo do reator proposto. A Figura 8.29 mostra a distribuição dessas regiões numa seção do núcleo do reator proposto obtidas com as contagens normalizadas de fissão. Como esperado, as regiões com maior contagem de fissão (Figura 8.26) coincidem com as áreas de maior liberação de energia, coincidentemente, de maior enriquecimento e ausente de isótopos de gadolínio.

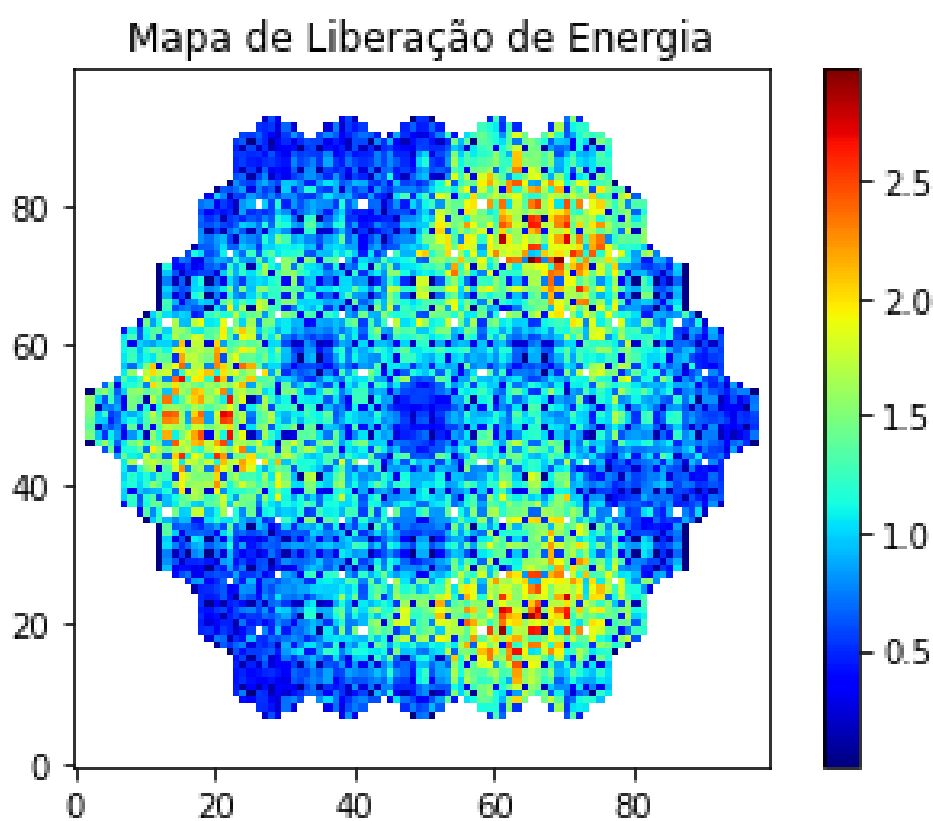


Figura 8.29: Mapa de liberação de energia no núcleo do reator proposto.

9 Conclusão

O OpenMC foi o código de transporte usado para simular o SMR deste trabalho. Trata-se de um *open source* mais recente que outros códigos já conhecidos e validados como MCNP e Serpent. Guardando similaridades com esses códigos, como o conceito de célula, universo etc, o OpenMC utiliza a linguagem orientada a objeto Python como API, cujos conceitos de objeto, classe e método são implementados na programação da simulação.

O benchmark BEAVRS foi simulado com o código de forma a avaliar sua performance com um modelo já conhecido. O benchmark consiste num PWR convencional de composição e geometria similares ao modelo comercial mais difundido desta tecnologia. O benchmark foi simulado e seu k_{eff} ficou abaixo de 1.01, classificado como levemente supercrítico, exatamente como a maioria dos reatores é projetada para operar, comprovando a capacidade do código em simular o comportamento de um modelo já conhecido. Depois da simulação do benchmark, as simulações do reator foram realizadas em seguida.

As simulações do reator baseado no modelo argentino CAREM 25 foram feitas e os valores estimados do fator de multiplicação efetivo, k_{eff} , via simulação direta e a partir de contagens derivadas foram comparados. O valor de k_{eff} resultante da simulação direta apresentou menor variância que o valor obtido de contagens derivadas, sendo mais recomendado (Tabela 8.4).

Também foi avaliada a influência dos parâmetros de simulação nos valores de k_{eff} . Em concordância com o Teorema Central do Limite (TCL), o maior número de partículas/ciclo resulta num valor mais preciso de k_{eff} , sendo 10000 partículas/ciclo o valor recomendado a ser setado no atributo `particle` no input dos parâmetros de simulação. A flutuação dos valores de k medidos durante a simulação é menor quanto maior este número, estabilizando a convergência do k_{eff} (Figura 8.1).

A fonte inicial preferencial foi a uniformemente distribuída, na forma de uma caixa de dimensões apropriadamente escolhida, cobrindo toda a zona fissil do núcleo e situada na origem dos eixos coordenadas arbitrariamente localizado no centro do reator, posicionando a fonte simetricamente no meio do núcleo. Este tipo de fonte começa a

estimar o fator de multiplicação efetivo de um valor mais próximo para o qual vai convergir que uma fonte pontual (Figura 8.3).

A distribuição de potência (regiões de maior liberação de energia) do reator é localizada em três áreas simetricamente dispostas e localizadas nas regiões de maior enriquecimento e sem a presença de veneno queimável (Gd_2O_3), (Figura 8.29).

A configuração dos enriquecimentos do reator resulta num sistema supercrítico, (k_{eff} aproximadamente 1.06), e cujo controle da reação em cadeia só pode ser feito com as barras de controle dos sistemas SA e SC inseridas no núcleo, devido à ausência de venenos naturais no início do ciclo.

A fuga de nêutrons (Leakage Fraction) do reator proposto é maior que a do benchmark devido ao menor inventário de água, que é o agente espalhador/refletor (Tabelas 7.4, 7.5 e 7.6). O benchmark, que é um PWR convencional, tem fuga de nêutrons praticamente nula. Sua enorme quantidade de água comparada à do reator proposto resulta numa economia de nêutrons muito maior (Tabelas 7.1, 7.2 e 7.3).

- Propostas de trabalhos futuros

- Simulação termohidráulica (**CFD**) do reator proposto;
- Simulação do reator com elemento combustível retangular, ao invés de hexagonal. Neste caso uma nova disposição das varetas com veneno queimável deverá ser definida;
- Simulação do reator utilizando outros tipos de veneno queimável que não baseados em gadolínio misturado ao combustível;
- Calculo de depleção isotópica;
- Determinação dos parâmetros cinéticos do reator.

ANEXO I

Os arquivos `materials.xml`, `geometry.xml`, `settings.xml` e `tallies.xml` do reator e do benchmark, assim como seus respectivos arquivos de simulação inteiros, estarão disponíveis num CD-ROM room que será depositado junto com a dissertação para os interessados.

Referências

- ALBUQUERQUE, F., Arriel, N., Beltrão, N. d. M. and Lucena, B. (2019), 'Ministério de minas e energia', *Boletim mensal dos* .
- ARENAZA, I. d. (2018), 'Videoconferencia sobre la central nuclear carem "primer reactor de potencia 100Videoconferencia sobre la Central Nuclear CAREM "Primer reactor de nuclear de potencia 100% argentino"a cargo del Ing. civil Ignacio de Arenaza. Jefe del Dpto. Ingeniería Civil y a/c de Ingeniería eléctrica. Gerencia Operativa/Gerencia de Área CAREM. Martes 30/10 a las 10hs. Palestra em espanhol disponível em : <https://www.youtube.com/watch?v=F0eLru-j5a4>. Acessado em: 18/11/2021.
- CAMARÃO, C., F. L. (2003), *Programação de Computadores em JAVA*, Editora LTC, Rio de Janeiro.
- CNEA (2017), 'Reactor carem', Disponível em : <https://www.argentina.gob.ar/buscar/Carem>. Acessado em: 17/09/2021.
- DA VEIGA, J. E. (2018), *Energia Nuclear: do anátema ao diálogo*, Senac.
- DE OLIVEIRA, A. M., Mario, M. C. and Pacheco, M. T. T. (2021), 'Fontes renováveis de energia elétrica: evolução da oferta de energia fotovoltaica no brasil até 2050', *Brazilian Applied Science Review* **5**(1), 257-272.
- DELMASTRO, D. (2021), 'Carem: Central argentina de elementos modulares', Palestra do Engenheiro Darío Delmastro, gerente de Engenharia do CAREM (CNEA), disponível em espanhol: <https://www.youtube.com/watch?v=TrdRkvh9u38&t=521s>. Acessado em: 11/06/2021.
- DELMASTRO, D. F., Gomez, S., Ishida, V., Mazzi, R., Santecchia, A. and Gomez de Soler, S. M. (2010), 'General aspects of carem-25 reactor'.
- DUDERSTADT, J. J. (1976), *Nuclear reactor analysis*, Wiley.
- EMPRESA DE PESQUISA ENERGÉTICA, E. (2020), *(PNE 2050)- Plano Nacional de Energia 2050*, Brasília.

- HORELIK, N., Herman, B., Ellis, M., Kumar, S., Liang, J., Forget, B. and Smith, K. (2020), *BEAVRS, Benchmark for Evaluation and Validation of Reactor Simulations*, MIT Computational Reactor Physics Group.
- LAMARSH, J. R. and BARATTA, A. J. (2001), *Introduction to nuclear engineering*, Vol. 3, Prentice hall Upper Saddle River, NJ.
- LIEBEROTH, J. (1968), “A Monte Carlo Technique to Solve the Static Eigenvalue Problem of the Boltzmann Transport Equation,” *Nukleonik*, 11, 213-219.
- MACHADO, L. and Hansen, G. L. (2018), ‘Opinião pública sobre energia nuclear enquanto sistema perito nas sociedades de risco da modernidade’, *Brazilian Journal of Radiation Sciences* **6**(3).
- MAGAN, H. B., Delmastro, D., Markiewicz, M., Lopasso, E., DIEZ, F., Gimenez, M., Rauschert, A., Halpert, S., Chocrón, M., Dezzutti, J. et al. (2011), ‘Carem project status’, *Science and Technology of Nuclear Installations* **2011**.
- MAGAN, H. B., Delmastro, D., Markiewicz, M., Lopasso, E., Diez, F., Giménez, M., Rauschert, A., Halpert, S., Chocrón, M., Dezzutti, J. et al. (2014), ‘Carem prototype construction and licensing status’, *IAEA, Vienna, Austria, IAEA-CN-164-5S01* .
- MARCEL, C. P., Delmastro, D. F., Schlamp, M. and Calzatta, O. (2017), ‘Carem-25: A safe innovative small nuclear power plant’.
- MARKIEWICZ, M. (2014), ‘Carem: Central argentina de elementos modulares’, Informações obtidas de M. Markiewicz, División de Elementos Combustibles, Centro Atómico de Bariloche, CNEA, em apresentação no Simpósio Small Modular Reactors for Nuclear Power, realizado no Rio de Janeiro, em Julho de 2014.
- MAZZI, R. (2005), ‘Carem: An innovative-integrated pwr’.
- MEDEIROS, E. S. (2004), *Desenvolvendo Software com UML 2.0*, Makron Books, São Paulo.
- MUELLER, J. P. (2019), *Programação Funcional Para Leigos*, Alta Books; 1ª edição.

- OPENMC-Contributors (2020), ‘Openmc documentation’, Disponível em <https://docs.openmc.org/en/stable> Acessado em: Novembro/2021.
- PRĀVĀLIE, R. and Bandoc, G. (2018), ‘Nuclear energy: Between global electricity demand, worldwide decarbonisation imperativeness, and planetary environmental implications’, *Journal of environmental management* **209**, 81–92.
- ROMANO, P. K. and Forget, B. (2013), ‘The openmc monte carlo particle transport code’, *Annals of Nuclear Energy* **51**, 274–281.
- ROMANO, P. K., Nicholas, E. H., Bryan, R. H., Adam, G. N., Forget, B. and SMITH, K. (2015), ‘Openmc: A state-of-the-art monte carlo code for research and development’, *Annals of Nuclear Energy* **82**, 90–97.
- ROWINSKI, M. K., White, T. J. and Zhao, J. (2015), ‘Small and medium sized reactors (smr): A review of technology’, *Renewable and Sustainable Energy Reviews* **44**, 643–656.
- SAIDI, K. and Omri, A. (2020), ‘Reducing co2 emissions in oecd countries: Do renewable and nuclear energy matter?’, *Progress in Nuclear Energy* **126**, 103425.
- SANTOS, R. (2003), *Introdução à Programação Orientada a Objetos Usando JAVA*, Editora Campus, Rio de Janeiro.
- SINTES, A. (2002), *Aprenda programação Orientada a Objetos em 21 dias*, Pearson, São Paulo.
- TASHAKOR, S., Zarifi, E. and Naminazari, M. (2017), ‘Neutronic simulation of carem-25 small modular reactor’, *Progress in Nuclear Energy* **99**, 185–195.
- VILLARINO, E., Hergenreder, D. and Matzkin, S. (2012), ‘Neutronic core performance of carem-25 reactor; comportamiento neutronico del nucleo del reactor carem-25’.
- VUOLO, J. H. (1996), *Fundamento da Teoria de Erros*, Editora Edgard Blücher.

WORLDNUCLEARASSOCIATION (2016), 'Nuclear power reactors', Disponível em <https://www.world-nuclear.org/information-library/nuclear-fuel-cycle/nuclear-power-reactors/nuclear-power-reactors.aspx>. Acessado em: Novembro/2021.