

**PROYECTO INTEGRADOR DE LA CARRERA DE
INGENIERÍA NUCLEAR**

**INCORPORACIÓN DE TÉCNICAS DE MUESTREO
MEDIANTE HISTOGRAMAS MULTIDIMENSIONALES
AL CÓDIGO DE SIMULACIÓN DE FUENTES DE
MONTE CARLO KDSOURCE**

**Lucas Ezequiel Ovando
Estudiante**

Ariel Marquez
Director

Zoe Prieto
Codirectora

Miembros del Jurado
Edmundo Lopasso
Norberto Schmidt

20 de Junio de 2025

Departamento de Física de Reactores y Radiaciones

Instituto Balseiro
Universidad Nacional de Cuyo
Comisión Nacional de Energía Atómica
Argentina

A todos por igual

Índice de símbolos

- API: Application Programming Interface.
- RAM: Random Access Memory.
- KDE: Kernel Density Estimation.
- OpenMC: código abierto Monte Carlo de transporte de partículas.
- KDSource: código abierto para generar fuentes distribucionales para simulaciones Monte Carlo.
- MCPL: formato de listas de partículas (Monte Carlo Particle List).
- XML: formato de archivo utilizado para definir fuentes en OpenMC/KDSource.
- On-the-fly: muestreo realizado dinámicamente durante la ejecución de una simulación.
- Bin: intervalo en el que se discretiza una variable para construir un histograma.
- Espacio de fases: conjunto de variables que definen el estado de una partícula.

Índice de contenidos

Índice de símbolos	v
Índice de contenidos	vii
Índice de figuras	xi
Índice de tablas	xvii
Resumen	xix
Abstract	xxi
1. Introducción	1
1.1. Contexto y motivación	1
1.2. Trabajos relacionados	5
1.3. Aportes específicos de este trabajo	6
2. Herramientas utilizadas	7
2.1. OpenMC	7
2.2. KDSource	8
2.3. Formatos de listas de partículas: MCPL y HDF5	8
2.4. Formato XML	8
2.5. Lenguajes de programación: Python, C y C++	9
3. Metodología para la generación de fuentes Monte Carlo mediante histogramas multidimensionales	13
3.1. Introducción general al método	13
3.2. Definición del espacio de fases ($E-r-\Omega$)	13
3.3. Procesamiento del archivo de partículas original	14
3.3.1. Preprocesamiento del archivo de partículas	14
3.3.2. Utilización de histogramas macro y micro	14
3.3.3. Métodos de discretización aplicados a histogramas macro y micro	16
3.4. Remuestreo de partículas en simulaciones Monte Carlo subsecuentes . .	17

3.5. Implementación computacional	19
4. Validación del método implementado: canal de vacío rodeado por agua	21
4.1. Descripción del modelo simulado en <code>OpenMC</code>	22
4.2. Descripción del archivo de partículas original	23
4.3. Procesamiento preliminar del archivo de partículas	26
4.3.1. Metodología del análisis	26
4.3.2. Distribuciones de letargía	28
4.3.3. Distribuciones de μ	30
4.3.4. Distribuciones espaciales X - Y	33
4.3.5. Análisis preliminar de los resultados	35
4.4. Procesamiento optimizado del archivo de partículas	36
4.5. Resultados de la simulación comparativa	39
4.5.1. Comparación del perfil de flujo a lo largo del canal	39
4.5.2. Comparación del espectro en la salida del canal	40
4.5.3. Comparación de la corriente	41
4.6. Conclusiones preliminares	43
5. Validación experimental: Conducto N°5 del reactor RA-6	45
5.1. Descripción del reactor RA-6 y instalación experimental del conducto N°5	46
5.2. Procesamiento del archivo de partículas	48
5.3. Resultados de simulación en <code>OpenMC</code> y comparación experimental	50
5.3.1. Distribución espacial del flujo	51
5.3.2. Comparación con la medición experimental	52
6. Resumen y conclusiones	57
A. Ejemplificación del <i>bineado</i> adaptativo	59
Caso con 1 bin	60
Caso con 2 bines	61
Caso con 3 bines	62
Caso con 5 bines	63
Caso con 10 bines	64
Caso con 25 bines	65
Caso con 100 bines	66
B. Implementación computacional del método de generación de fuentes Monte Carlo mediante histogramas multidimensionales	69
B.1. Procesamiento de un archivo HDF5 en Python	69

B.1.1. Estructura <code>TreeNode</code>	70
B.1.2. Construcción recursiva del árbol	73
B.2. <i>Bineado</i> adaptativo y generación del XML	78
B.2.1. Función de <i>bineado</i> adaptativo en Python	78
B.2.2. Ejemplo de archivo XML generado	81
B.3. Implementación de <code>HistogramSource</code> en C/C++ para <code>OpenMC</code>	83
B.3.1. Definición de <code>HistogramSource</code> en <code>source.h</code>	83
B.3.2. Función <code>fill_particle</code> y <code>traverse</code> (<code>histograms.c</code>)	85
B.3.3. Inicialización de la fuente <i>on-the-fly</i> desde la API de <code>OpenMC</code>	92
B.4. Ejemplo de flujo completo	92
Bibliografía	95
Agradecimientos	97

Índice de figuras

1.1.	Representación esquemática del núcleo de un reactor tipo piletas con canal de extracción. Se observa el núcleo central, rodeado por agua y blindaje biológico, así como un canal de extracción de neutrones. El gradiente de color indica cualitativamente la distribución de la población neutrónica típica en simulaciones Monte Carlo desde el núcleo.	2
1.2.	Esquema ilustrativo del proceso de desacople geométrico en simulaciones Monte Carlo para un reactor de investigación tipo piletas. En la primera etapa se registra un archivo de partículas en la entrada del canal de extracción, el cual es posteriormente utilizado para generar una fuente distribucional que permite simular eficientemente el canal de forma independiente. El gradiente de colores representa la disminución de la población neutrónica.	4
2.1.	Diagrama de soporte de MCPL.	9
2.2.	Etapa de detección: registro de partículas en formato MCPL a partir de una simulación en OpenMC. Bloques con esquinas redondeadas representan procesos computacionales, mientras que los bloques rectangulares indican archivos que se registran o leen desde disco.	10
2.3.	Etapa de procesamiento: construcción de una fuente distribucional a partir de un archivo MCPL. Bloques con esquinas redondeadas representan procesos computacionales, mientras que los bloques rectangulares indican archivos que se registran o leen desde disco.	11
2.4.	Etapa de producción: uso del archivo XML para generar nuevas partículas de forma offline o on-the-fly. Bloques con esquinas redondeadas representan procesos computacionales, mientras que los bloques rectangulares indican archivos que se registran o leen desde disco.	11
3.1.	Representación del espacio de fases con las variables $(E, x, y, z, \theta, \phi)$, donde E es la energía, x , y y z son coordenadas espaciales y θ y ϕ son las variables direccionales.	14

3.2. Esquema ilustrativo de la estructura jerárquica de macrogrupos y microgrupos formando un árbol. Se resalta la jerarquía entre variables.	16
3.3. Esquema ilustrativo del proceso secuencial de generación de partículas a partir de la distribución jerárquica guardada, resaltando el muestreo sucesivo mediante números pseudoaleatorios.	18
4.1. Esquema del proceso de remuestreo y validación del método implementado.	22
4.2. Esquema del modelo simulado en <code>OpenMC</code>	23
4.3. Distribución de X vs Y para el archivo de partículas registrado en la primera simulación del tubo de vacío.	24
4.4. Distribuciones de letargía y μ para el archivo de partículas registrado en la primera simulación del tubo de vacío, con escala logarítmica en el eje y . Se observa la presencia de dos poblaciones: una concentrada en $\mu = 1$ y letargía fija correspondiente a una energía de $E = 1 \text{ MeV}$, montada sobre una distribución continua de neutrones colisionados.	25
4.5. Distribuciones de letargía vs μ para el archivo de partículas original. Se observa la presencia de un conjunto de neutrones graficados como un píxel en $\mu = 1$ y letargía correspondiente a $E = 1 \text{ MeV}$. A su vez se observa la presencia de un segundo conjunto de neutrones colisionados, que se distribuyen en el plano de forma continua, con un agrupamiento sobre letargía = 20, indicando la termalización de los neutrones.	26
4.6. Distribución remuestreada de letargía para el caso de <i>bineado</i> uniforme sin bordes manuales.	29
4.7. Distribución remuestreada de letargía para el caso de <i>bineado</i> uniforme con borde manual en la zona de letargía correspondiente a una energía de $E = 1 \text{ MeV}$	29
4.8. Distribución remuestreada de letargía para el caso de <i>bineado</i> adaptativo sin bordes manuales.	30
4.9. Distribución remuestreada de letargía para el caso de <i>bineado</i> adaptativo con borde manual en la zona de letargía correspondiente a una energía de $E = 1 \text{ MeV}$	30
4.10. Distribución remuestreada de μ para el caso 1 de <i>bineado</i> uniforme sin bordes manuales.	31
4.11. Distribución remuestreada de μ para el caso 2 de <i>bineado</i> uniforme con borde manual en $\mu \approx 1$	32
4.12. Distribución remuestreada de μ para el caso 3 de <i>bineado</i> adaptativo sin bordes manuales.	32
4.13. Distribución remuestreada de μ para el caso 4 de <i>bineado</i> adaptativo con borde manual en $\mu \approx 1$	33

4.14. Distribución remuestreada en el plano $X-Y$ para el caso 1 de <i>bineado</i> uniforme sin bordes manuales.	34
4.15. Distribución remuestreada en el plano $X-Y$ para el caso 2 de <i>bineado</i> uniforme con bordes manuales en la interfaz agua–vacío.	34
4.16. Distribución remuestreada en el plano $X-Y$ para el caso 3 de <i>bineado</i> adaptativo sin bordes manuales.	35
4.17. Distribución remuestreada en el plano $X-Y$ para el caso 4 de <i>bineado</i> adaptativo con bordes definidos por quien utiliza el código.	36
4.18. Distribución remuestreada de letargía utilizando la configuración optimizada de macro y microgrupos.	37
4.19. Distribución remuestreada de μ utilizando la configuración optimizada.	38
4.20. Distribución remuestreada en el plano $X-Y$ con configuración final optimizada.	38
4.21. Comparación entre los perfiles de flujo escalar obtenidos mediante la fuente original (simulación larga desde el inicio del conducto) y mediante la fuente distribucional (simulación desde la superficie intermedia). En los casos donde no se aprecia las barras de error se debe a que son menores que el tamaño del símbolo. En ambas regiones se muestra también el error relativo porcentual.	40
4.22. Comparación de la distribución de letargía normalizada por neutrón de fuente (S_0) en la superficie a $z = 80$ cm para las simulaciones original y remuestreada. En ambos casos se incluye también el error relativo porcentual.	41
5.1. Esquema representativo del núcleo del RA-6. En el mismo se observa la disposición de los elementos combustibles, el conducto N°1 (inferior) y el conducto N°5 (superior).	46
5.2. Esquema representativo del conducto N°5 del RA-6 y de la instalación experimental asociada al <i>chopper</i>	47
5.3. Esquema representativo del disco rotatorio del <i>chopper</i> empleado en el conducto N°5. La ranura permite el paso periódico de neutrones, generando un haz pulsado.	48
5.4. Distribución espacial de las partículas del archivo original en el plano XY	49
5.5. Distribución de letargía de las partículas del archivo original.	50
5.6. Distribución espacial de las partículas de la fuente distribucional en el plano XY	51
5.7. Comparación de la distribución de letargía entre el archivo original y la fuente distribucional.	52

5.8. Distribución del flujo neutrónico en el modelo del conducto N°5.	53
5.9. Comparación del espectro de neutrones obtenido en el banco de detectores en la simulación de OpenMC con la medición experimental realizada por TdV por el Departamento de Neutrones del CAB. Para ambas curvas se grafican los errores estadísticos.	54
A.1. Distribución de letargía utilizada en la exemplificación.	59
A.2. Distribución de letargía aproximada con un único bin. Se observa que el histograma es constante, representando el valor promedio sobre todo el dominio.	60
A.3. Evaluación del criterio de refinamiento para el caso de 1 bin. Se observa que la máxima diferencia ocurre cerca de letargía ~ 3 , lo que indica que el nuevo bin debe ser ubicado en esa posición.	61
A.4. Distribución de letargía aproximada con 2 bines. Se observa que el método comienza a capturar la delta en 1 MeV, pero con un bin de ancho considerable.	61
A.5. Evaluación del criterio de refinamiento para el caso de 2 bines. Se observa que la máxima diferencia ocurre nuevamente cerca de letargía ~ 3 , lo que indica que el nuevo bin debe ser ubicado en esa posición.	62
A.6. Distribución de letargía aproximada con 3 bines. Se observa que el método logra capturar la delta en 1 MeV.	62
A.7. Evaluación del criterio de refinamiento para el caso de 3 bines. Se observa que la máxima diferencia ocurre cerca de letargía ~ 18 y letargía ~ 22 , lo que indica que los próximos dos bines deben ser ubicados en esas posiciones.	63
A.8. Distribución de letargía aproximada con 5 bines. Se observa que el método comienza a capturar la distribución en la región de termalización.	63
A.9. Evaluación del criterio de refinamiento para el caso de 5 bines. Se observa en la diferencia absoluta entre FDCs los valores de mayor discrepancia, que indican que los próximos bines deben ser ubicados en esas posiciones.	64
A.10. Distribución de letargía aproximada con 10 bines. Se observa que el método mejora la aproximación de la distribución en la región de termalización y en la región epitérmica.	64
A.11. Evaluación del criterio de refinamiento para el caso de 10 bines. Se observa en la diferencia absoluta entre FDCs los valores de mayor discrepancia, que indican que los próximos bines deben ser ubicados en esas posiciones.	65

A.12.Distribución de letargía aproximada con 25 bines. Se observa que el método logra capturar la distribución de letargía, salvo en la región de letargía mayor a 22, debido a la poca intensidad que tiene la distribución en esa región.	65
A.13.Evaluación del criterio de refinamiento para el caso de 25 bines. Se observa en la diferencia absoluta entre FDCs los valores de mayor discrepancia, que indican que los próximos bines deben ser ubicados en esas posiciones.	66
A.14.Distribución de letargía aproximada con 100 bines. Se observa que el método logra capturar con precisión la distribución de letargía, salvo en la región de letargía mayor a 22, debido a la poca intensidad que tiene la distribución en esa región.	66
A.15.Evaluación del criterio de refinamiento para el caso de 100 bines.	67

Índice de tablas

Resumen

En este trabajo se implementó un método alternativo al *Kernel Density Estimation* (KDE) para la generación de fuentes distribucionales en simulaciones Monte Carlo, empleando histogramas multidimensionales dentro del entorno de los códigos abiertos `KDSouce` y `OpenMC`. La metodología propuesta implica el procesamiento jerárquico utilizando histogramas multidimensionales para estimar distribuciones de partículas, garantizando la preservación de correlaciones en el espacio de fases.

Se desarrolló un esquema de *bineado* adaptativo, capaz de captar regiones con cambios bruscos en las distribuciones, logrando así mejorar la resolución local del histograma. La validación del método se llevó a cabo mediante la comparación entre simulaciones y datos experimentales obtenidos previamente por el Departamento de Neutrones del Centro Atómico Bariloche, en el conducto N°5 del reactor RA-6. Los resultados mostraron una adecuada reproducción de los datos experimentales, verificando la efectividad del método implementado.

Palabras clave: MONTE CARLO, HISTOGRAMAS MULTIDIMENSIONALES, OPENMC, KDSOURCE, RA-6, INSTITUTO BALSEIRO

Abstract

In this work, an alternative method to *Kernel Density Estimation* (KDE) was implemented for generating distributional sources in Monte Carlo simulations, employing multidimensional histograms within the open-source code frameworks `KDSOURCE` and `OpenMC` frameworks. The proposed methodology involves hierarchical processing using multidimensional histograms to estimate particle distributions, preserving the correlations among phase space.

An adaptive binning scheme was developed to effectively capture regions with abrupt changes in the distributions, thereby enhancing local histogram resolution. Validation of the method was performed by comparing simulation results with experimental data previously obtained by the Neutron Department of *Centro Atómico Bariloche*, in beam N°5 of the RA-6 research reactor. Results demonstrated good agreement with the experimental data, confirming the effectiveness of the implemented method.

Keywords: MONTE CARLO, MULTIDIMENSIONAL HISTOGRAMS, OPENMC, KDSOURCE, RA-6, INSTITUTO BALSEIRO

Capítulo 1

Introducción

1.1. Contexto y motivación

Las simulaciones Monte Carlo son la herramienta estándar cuando la complejidad geométrica o la marcada anisotropía angular del flujo neutrónico dificultan la aplicación de métodos determinísticos. Sin embargo, cuando las partículas atraviesan blindajes o regiones altamente absorbentes, la estadística disponible en la zona de interés se reduce considerablemente, lo que incrementa la incertidumbre en magnitudes físicas clave como el flujo escalar o la dosis equivalente ambiental, e incluso puede imposibilitar su cálculo debido a nula estadística.

En este trabajo abordaremos la temática de la obtención de resultados a la salida de canales de extracción de neutrones en reactores de investigación tipo piletas. En estos sistemas, los neutrones generados en el núcleo deben atravesar el agua de la piletita antes de alcanzar la entrada del canal de extracción. Durante una simulación desde el núcleo, la mayoría de los neutrones permanecen dentro del mismo y aquellos que logran salir al agua son en gran medida absorbidos a medida que penetran en esta. En consecuencia, son escasos los neutrones que finalmente alcanzan la entrada del canal, haciendo que resulte computacionalmente costoso obtener estadística suficiente a la salida del canal, en el ambiente circundante o en dispositivos experimentales.

En la Figura 1.1 se presenta un esquema ilustrativo del núcleo de un reactor de investigación tipo piletita con un canal de extracción de neutrones. Este esquema consta de un núcleo central rodeado por agua y rodeado externamente por un blindaje biológico. El canal de extracción permite obtener un haz de neutrones destinado a usos posteriores en investigaciones experimentales. Además, se muestra esquemáticamente mediante un gradiente de color las regiones con mayor estadística neutrónica y cómo esta disminuye drásticamente al avanzar por el agua y el blindaje. Este esquema ejemplifica una simulación típica de núcleo, ilustrando claramente las dificultades que se tienen para obtener estadística lejos del núcleo.

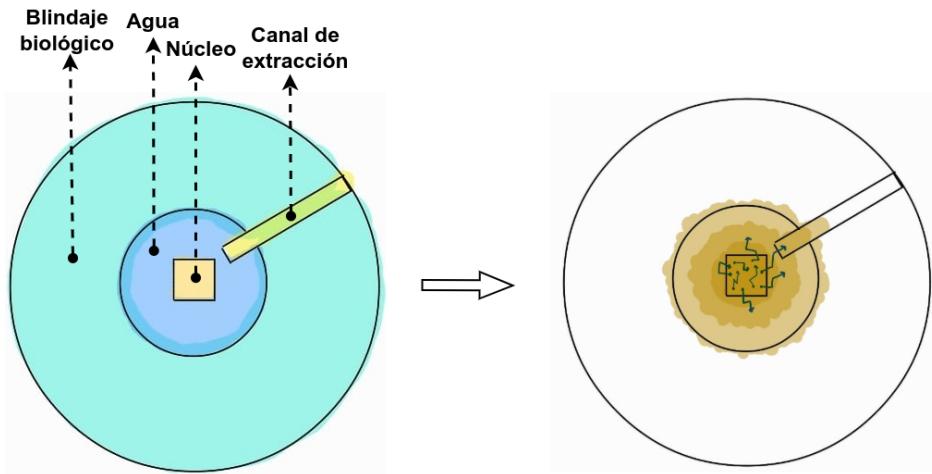


Figura 1.1: Representación esquemática del núcleo de un reactor tipo piletas con canal de extracción. Se observa el núcleo central, rodeado por agua y blindaje biológico, así como un canal de extracción de neutrones. El gradiente de color indica cualitativamente la distribución de la población neutrónica típica en simulaciones Monte Carlo desde el núcleo.

Para mitigar esta problemática, se suelen emplear técnicas de reducción de varianza tales como:

- **Separación geométrica del problema:** consiste en dividir la simulación en regiones contiguas, separando la fuente original de la región de interés mediante una superficie de interfaz. En dicha superficie se registran las variables del espacio de fases y el peso estadístico de las partículas que la atraviesan, con el objetivo de construir una nueva fuente de partículas estadísticamente equivalente. Esta nueva fuente permite reiniciar la simulación a partir de dicha superficie, concentrando los recursos computacionales exclusivamente en la región de interés. Existen diversos métodos para modelar este tipo de fuentes distribucionales a partir del archivo de partículas registrado, tales como el ajuste de funciones unidimensionales o multidimensionales, el método de *smearing* utilizado en *McStas* [1], la estimación por densidad de kernels (*Kernel Density Estimation*, KDE) adaptativa [2], o el uso de histogramas multidimensionales [3].
- **Absorción implícita:** en lugar de eliminar partículas cuando son absorbidas, se les reduce su peso estadístico de acuerdo a la probabilidad de absorción, permitiendo que las partículas sobrevivan y continúen su historia.
- **Ruleta rusa:** se aplica para eliminar partículas con bajo peso estadístico, preservando el valor esperado del cálculo. Esta técnica se utiliza típicamente en conjunto con la absorción implícita, ya que sin ella todas las partículas mantienen peso unitario y la aplicación de ruleta rusa no sería necesaria. Cada partícula con peso inferior a un umbral predefinido tiene una cierta probabilidad de ser eliminada; si sobrevive, su peso es incrementado.

- **Ventanas de peso:** esta técnica define un rango aceptable de pesos estadísticos para las partículas en cada región del dominio. Aquellas con peso muy alto son divididas (*splitting*) y las de peso muy bajo se someten a ruleta rusa para controlar la varianza espacialmente.

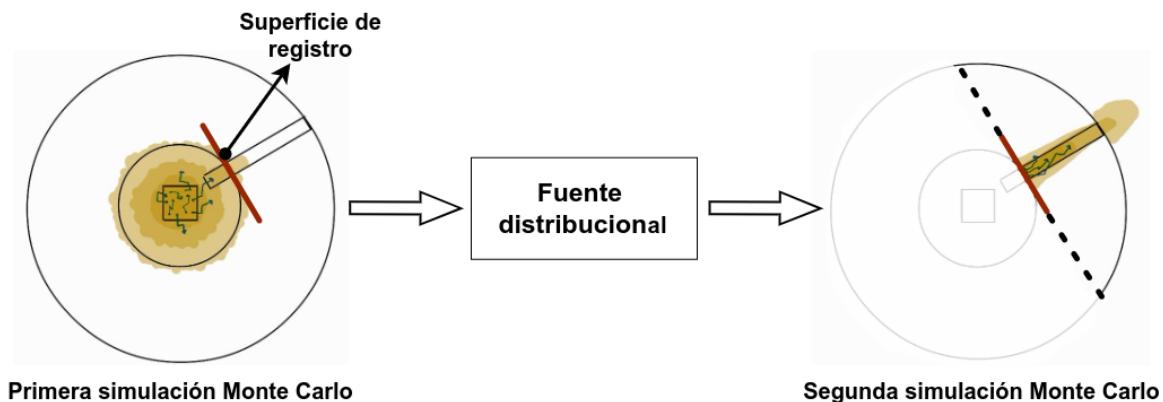
El método desarrollado en este trabajo se basa en la separación geométrica del problema en dos regiones: una que contiene el núcleo del reactor, y otra que abarca exclusivamente la región de interés. Esta técnica se implementa mediante la división del modelo simulado en dos regiones, delimitadas por superficies de acople ubicadas estratégicamente dentro de la geometría. En una primera etapa, se realiza una simulación completa del núcleo hasta una superficie intermedia, donde se registran en una lista las propiedades de las partículas que la atraviesan, incluyendo su energía E , posición \mathbf{r} , dirección Ω , peso estadístico, y tipo de partícula (por ejemplo, neutrón o fotón). A partir de esta lista de partículas, se estima la distribución multidimensional en el espacio de fases utilizando histogramas multidimensionales. Esta estructura permite segmentar el conjunto original de datos mediante histogramas de baja resolución, llamados histogramas macro, que separan el espacio de fases en regiones donde las correlaciones entre variables se mantienen aproximadamente constantes. Sobre cada una de estas regiones, se construyen histogramas de mayor resolución —los histogramas micro— que aproximan la distribución de cada variable con mayor detalle. De este modo, se preservan tanto la forma general de la distribución como las correlaciones entre las variables registradas. A partir de esta estimación, es posible generar nuevas partículas que pertenezcan estadísticamente al mismo espacio de fases que la muestra original, obteniendo así una fuente sintética denominada fuente distribucional. Esta metodología se desarrolla en profundidad en el Capítulo 3, donde se describe el procedimiento implementado para su construcción.

En la Figura 1.2 se ejemplifica el proceso desarrollado para una simulación del núcleo de un reactor de investigación tipo piletta. Inicialmente, se realiza una simulación completa desde el núcleo, obteniendo una estadística adecuada en sus alrededores, la cual se reduce considerablemente a medida que los neutrones penetran en el agua circundante. Este fenómeno se visualiza mediante un gradiente de colores. En esta simulación se define una superficie de registro en la entrada de un canal de extracción, y las partículas que la atraviesan se registran en un archivo. A partir de dicho archivo, se construye una fuente distribucional mediante la metodología desarrollada en este trabajo. Esta fuente permite simular únicamente el canal, concentrando la estadística en esa región sin necesidad de simular desde el núcleo. En esta segunda simulación, la región previa a la superficie de interfaz se reemplaza completamente por vacío, dejando en el modelo únicamente la región de interés. Esto se logra definiendo una condición de vacío en la superficie de interfaz. Cabe destacar que, si bien podría existir la posibilidad

de que alguna partícula inicialmente dirigida hacia la región de la fuente reingrese a la región de interés por retrodispersión, dicho fenómeno ya está estadísticamente incorporado en la lista original de partículas registrada en la primera simulación.

En resumen, el procedimiento consta de tres etapas fundamentales:

- **Detección:** Registro de las variables del espacio de fases y del peso estadístico de cada partícula que atraviesa una superficie intermedia de desacople. El resultado de este proceso es un archivo de partículas que contiene la información necesaria para caracterizar la distribución de la fuente sobre dicha superficie.
- **Estimación:** Procesamiento del archivo de partículas mediante la metodología propuesta en este trabajo, basada en histogramas multidimensionales. Este procedimiento permite aproximar la distribución en el espacio de fases, preservando las correlaciones entre variables, y genera como resultado una fuente distribucional.
- **Producción:** Utilización de la fuente distribucional estimada para generar nuevas partículas que pertenezcan al espacio de fases del archivo de partículas original. Estas partículas son empleadas en simulaciones posteriores, permitiendo modelar la región de interés de forma desacoplada de la fuente original.



Primera simulación Monte Carlo

Segunda simulación Monte Carlo

Figura 1.2: Esquema ilustrativo del proceso de desacople geométrico en simulaciones Monte Carlo para un reactor de investigación tipo piletta. En la primera etapa se registra un archivo de partículas en la entrada del canal de extracción, el cual es posteriormente utilizado para generar una fuente distribucional que permite simular eficientemente el canal de forma independiente. El gradiente de colores representa la disminución de la población neutrónica.

El archivo original de partículas contiene inevitablemente regiones del espacio de fases con baja estadística o sin eventos registrados, producto de su carácter finito. Si se reutiliza directamente como fuente en una simulación posterior —por ejemplo, simulando varias veces las mismas partículas registradas— se corre el riesgo de reproducir el mismo ruido estadístico, afectando la calidad de los resultados.

1.2. Trabajos relacionados

Los trabajos previos realizados en el Departamento de Física de Reactores y Radiaciones (DeFRRa) del Centro Atómico Bariloche (CAB) [4] abordaron el modelado de fuentes distribucionales mediante el uso de histogramas multidimensionales [3, 5, 6]. Esta estrategia permitió capturar correlaciones entre energía, posición y dirección en geometrías planas rectangulares, siendo aplicada satisfactoriamente en estudios vinculados al reactor RA-10.

En continuidad con dichos desarrollos, la herramienta `KDSouce` —desarrollada en conjunto por la Comisión Nacional de Energía Atómica (CNEA) [7] y por el Instituto Balseiro (IB) [8]— introdujo técnicas más avanzadas basadas en *Kernel Density Estimation (KDE)*, en forma multivariable y adaptativa [2, 9–11]. `KDSouce` permite ajustar automáticamente distribuciones continuas a partir de archivos de partículas previamente obtenidos y, a partir de ellas generar fuentes distribucionales, preservando las correlaciones del espacio de fases. Estas fuentes pueden ser utilizadas directamente en simulaciones Monte Carlo mediante el código `OpenMC` [12].

No obstante, el enfoque basado en *KDE* presenta una limitación: su carácter inherentemente suavizante puede dificultar la representación precisa de discontinuidades abruptas en el espacio de fases. En particular, el método puede fallar al capturar regiones donde las distribuciones presentan derivadas segundas muy marcadas o donde ocurren cambios bruscos, como bordes definidos por condiciones geométricas o materiales. En estos casos, la suavización de las distribuciones multivariadas puede degradar la calidad de la fuente generada, especialmente cuando se busca preservar estructuras finas o anisotropías marcadas.

El presente trabajo extiende estas líneas incorporando los enfoques basados en histogramas multidimensionales dentro del entorno ya establecido de `KDSouce`, proponiendo además un método de discretización adaptativa que automatiza la definición de las grillas de los histogramas macro y micro. A diferencia de los trabajos anteriores, este nuevo enfoque permite segmentaciones variables para diferentes subconjuntos del espacio de fases. Por ejemplo, el método puede asignar una discretización angular más refinada para neutrones rápidos, donde suelen prevalecer anisotropías asociadas a haces colimados, y una discretización más homogénea para neutrones térmicos, cuya distribución tiende a ser más isotrópica. De esta forma, se logra preservar las correlaciones relevantes sin imponer una malla uniforme en todo el dominio, mejorando la capacidad de representación del método.

1.3. Aportes específicos de este trabajo

Este trabajo busca profundizar el desarrollo de `KDSource` mediante la incorporación de histogramas multidimensionales como una alternativa -o complemento- a la metodología *KDE* existente. Las contribuciones específicas son:

- Implementación de histogramas multidimensionales en `KDSource`, capaces de:
 - preservar las correlaciones entre variables ($\mathbf{E}-\mathbf{r}-\Omega$) para fuentes definidas sobre un plano;
 - representar adecuadamente discontinuidades o picos en todas las variables;
 - implementar un método de selección automática y adaptativa de bordes para los histogramas, tanto a nivel macro como micro, optimizado según la estadística disponible en cada subgrupo del espacio de fases.
- Integración optimizada del flujo de trabajo en `OpenMC` mediante:
 - generación *offline* de un archivo intermedio en formato XML contenido la fuente distribucional, que incluye la información de los histogramas multidimensionales;
 - desarrollo de un módulo en C que utilice eficientemente esta información para producir partículas individualmente;
 - implementación de un remuestreo *on-the-fly* integrado en `OpenMC`, minimizando la ocupación de memoria al evitar la carga y gestión de archivos extensos de partículas.
- Validación del método en casos de complejidad creciente:
 - Un caso simplificado, consistente en un haz colimado ingresando en un paralelepípedo de agua atravesado por un canal de vacío, con fuentes definidas artificialmente para permitir una comparación con una simulación directa más extensa sin la aplicación del método desarrollado.
 - Un caso real, consistente en la propagación a través del conducto N°5 del reactor RA-6, utilizando un archivo de partículas proporcionado por el departamento de Física de Neutrones del CAB, que fue generado a través de una simulación del núcleo en `OpenMC`.

Capítulo 2

Herramientas utilizadas

Este capítulo tiene por objetivo presentar y describir las herramientas computacionales utilizadas a lo largo de este trabajo. Específicamente, se abordará el programa `OpenMC`, los formatos de archivos usados (`MCPL` y `XML`), y los lenguajes de programación involucrados (`Python`, `C` y `C++`).

2.1. `OpenMC`

`OpenMC` es un programa de simulación Monte Carlo desarrollado inicialmente en el *Massachusetts Institute of Technology* como *software* de código abierto [12]. Actualmente es mantenido por el *Argonne National Laboratory* junto a una activa comunidad internacional que contribuye continuamente a su desarrollo y expansión. Está especialmente orientado al cálculo del transporte de neutrones y fotones, permitiendo tanto simulaciones de criticidad como simulaciones de fuente fija. En este trabajo se emplean exclusivamente simulaciones de fuente fija.

Una de las ventajas de `OpenMC` es su flexibilidad en la obtención de resultados. El código permite registrar diferentes magnitudes físicas como flujos, dosis, corrientes, espectros energéticos, etc., además de ofrecer la posibilidad de registrar partículas que atraviesan superficies definidas por quien utiliza el programa en un archivo de formato `HDF5` [13] o `MCPL` [14]. Estos archivos almacenan información de las variables de las partículas, permitiendo su posterior análisis o reutilización para generar nuevas simulaciones. Asimismo, `OpenMC` incorpora técnicas avanzadas de reducción de varianza, como las ventanas de peso (*weight windows*), que resultan fundamentales para reducir la incertidumbre estadística de los resultados en áreas de interés específico, técnica utilizada en este trabajo.

2.2. KDSource

`KDSource` es una herramienta computacional desarrollada en conjunto por CNEA e IB, cuyo objetivo principal es procesar archivos de partículas generados en simulaciones Monte Carlo para la construcción de nuevas fuentes distribucionales [15]. Su integración con `OpenMC` permite desacoplar geométricamente simulaciones complejas, facilitando significativamente el cálculo de transporte en geometrías difíciles o extensas.

El fundamento original de `KDSource` reside en la técnica **Kernel Density Estimation (KDE)**, una técnica estadística que permite estimar distribuciones continuas de variables a partir de muestras discretas, manteniendo la correlación existente entre ellas.

2.3. Formatos de listas de partículas: MCPL y HDF5

En simulaciones Monte Carlo, es común registrar partículas que atraviesan una superficie o ingresan a una región de interés, generando lo que se conoce como una lista de partículas. Estos archivos permiten capturar el estado de cada partícula —incluyendo su energía, posición, dirección, peso estadístico y tipo de partícula— al momento de cruzar una superficie.

`OpenMC` emplea por defecto el formato `HDF5` para almacenar estas listas [13]. Este formato binario jerárquico permite almacenar datos de manera eficiente, estructurada y accesible desde diversos lenguajes de programación. Sin embargo, su estructura está adaptada específicamente al ecosistema de `OpenMC`, dificultando su reutilización directa en otros códigos de transporte.

Con el objetivo de facilitar la interoperabilidad entre distintos códigos Monte Carlo, existe el formato **MCPL (Monte Carlo Particle List)** [14]. Este estándar de listas de partículas permite almacenar listas generadas en simulaciones de forma compacta y eficiente, preservando información esencial como la energía, posición, dirección, peso de cada partícula y tipo de partícula. A diferencia de formatos específicos como `HDF5`, `MCPL` fue concebido como una interfaz entre diferentes entornos Monte Carlo.

La Figura 2.1 muestra los diversos códigos que pueden producir o consumir archivos en formato `MCPL`. En este proyecto, su uso permite vincular eficientemente los archivos de partículas generados por `OpenMC` con la herramienta `KDSource`.

2.4. Formato XML

El formato **XML (Extensible Markup Language)** es un lenguaje utilizado para describir y almacenar información estructurada de manera jerárquica, basándose en una organización de datos tipo árbol [16]. Una de sus principales ventajas es que está

diseñado para ser tanto legible por máquinas como por humanos, lo que facilita su edición, inspección y validación manual durante el desarrollo. Debido a estas características, resulta particularmente adecuado para guardar las distribuciones estimadas mediante histogramas multidimensionales. A su vez, OpenMC emplea archivos XML para almacenar la configuración completa de sus simulaciones (geometría, materiales, configuración de *tallies*, etc.), por lo que la elección de este formato contribuye a una integración directa y eficiente entre los componentes desarrollados.

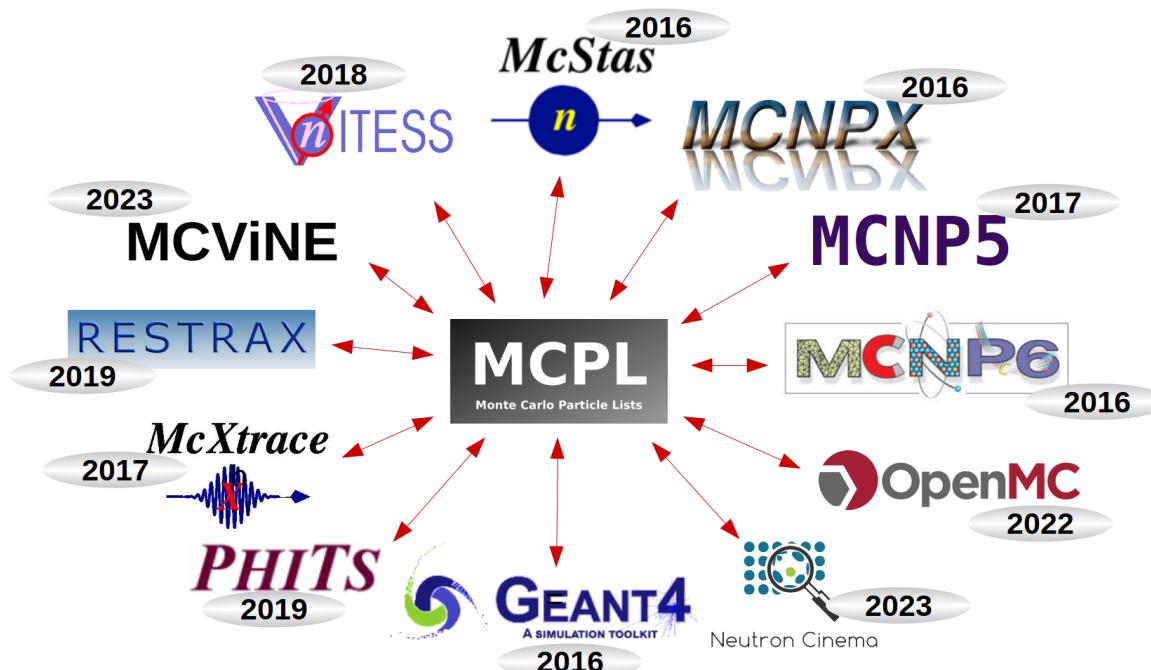


Figura 2.1: Diagrama de soporte de MCPL. Reproducido de [14].

2.5. Lenguajes de programación: Python, C y C++

A lo largo del desarrollo del proyecto, se utilizaron principalmente tres lenguajes de programación:

- **Python**: lenguaje de alto nivel especialmente adecuado para interfaces de usuario, análisis exploratorio de datos y configuración de simulaciones debido a su simplicidad, claridad y flexibilidad. KDSource hace uso de Python para la preparación y procesamiento de datos previos al remuestreo Monte Carlo. Tanto KDSource como OpenMC disponen de una interfaz en Python.
- **C**: lenguaje de programación de bajo nivel conocido por su eficiencia computacional, velocidad y control preciso sobre la gestión de memoria. Se utilizó en este trabajo para desarrollar módulos específicos encargados del remuestreo eficiente de partículas, especialmente cuando se requieren grandes volúmenes de datos. El módulo de remuestreo de KDSource está escrito en este lenguaje.

- **C++:** lenguaje de programación que extiende a C con capacidades de programación orientada a objetos. Esta arquitectura permite la expansión modular del código y facilita su mantenimiento. OpenMC está desarrollado en este lenguaje.

El desarrollo realizado en este trabajo implementa un flujo computacional para el uso de fuentes distribucionales en simulaciones Monte Carlo. Este flujo se compone de tres etapas principales: detección, procesamiento y producción, y aprovecha las capacidades de los lenguajes Python, C y C++.

Detección: se realiza mediante una simulación inicial en OpenMC, en la cual se registra un archivo de partículas sobre una superficie. Este archivo puede guardarse en formato HDF5, que es el formato nativo de OpenMC, y luego ser convertido al formato MCPL si se desea continuar con el procesamiento. Alternativamente, si OpenMC fue compilado con soporte para MCPL, puede generarse directamente en dicho formato. El archivo resultante se almacena en disco para su uso posterior. Este procedimiento se ilustra en la Figura 2.2.

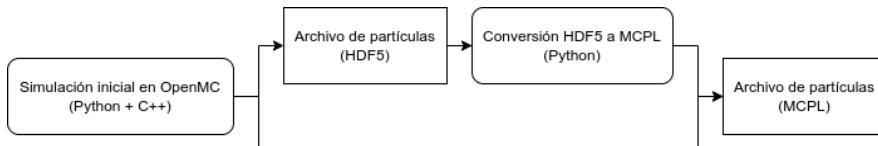


Figura 2.2: Etapa de detección: registro de partículas en formato MCPL a partir de una simulación en OpenMC. Bloques con esquinas redondeadas representan procesos computacionales, mientras que los bloques rectangulares indican archivos que se registran o leen desde disco.

Procesamiento: esta etapa se lleva a cabo en Python utilizando la implementación desarrollada para procesar mediante histogramas multidimensionales. A partir del archivo MCPL, se construye una fuente distribucional que representa la densidad de probabilidad estimada en el espacio de fases. El resultado de esta etapa es un archivo XML, que contiene la parametrización de la fuente y se almacena en disco como entrada para la etapa de producción. Este archivo XML constituye toda la información necesaria para generar nuevas partículas en simulaciones posteriores, sin requerir el acceso al archivo original de partículas. Esta característica representa una ventaja frente al método utilizado en KDSource, el cual necesita mantener el archivo original disponible en memoria durante la simulación posterior. Sin embargo, ambos métodos necesitan cargar en memoria el archivo de partículas original para el procesamiento. En casos donde dicho archivo posea un volumen considerable pueden derivar en un consumo elevado de memoria RAM. La Figura 2.3 muestra un esquema de esta etapa.

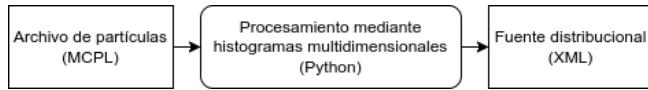


Figura 2.3: Etapa de procesamiento: construcción de una fuente distribucional a partir de un archivo MCPL. Bloques con esquinas redondeadas representan procesos computacionales, mientras que los bloques rectangulares indican archivos que se registran o leen desde disco.

Producción: esta etapa puede abordarse de dos maneras, implementada en código C. En la modalidad *offline*, el archivo XML es utilizado para generar una nueva lista de partículas en formato MCPL, que luego puede emplearse directamente como fuente en OpenMC si se cuenta con soporte para este formato. En caso contrario, puede convertirse nuevamente a HDF5. En la modalidad *on-the-fly*, se evita por completo el uso de listas intermedias: OpenMC accede directamente al archivo XML y realiza el remuestreo dinámicamente durante la simulación, gracias a una extensión *ad-hoc* incorporada en este trabajo. Ambas modalidades se ilustran en la Figura 2.4.

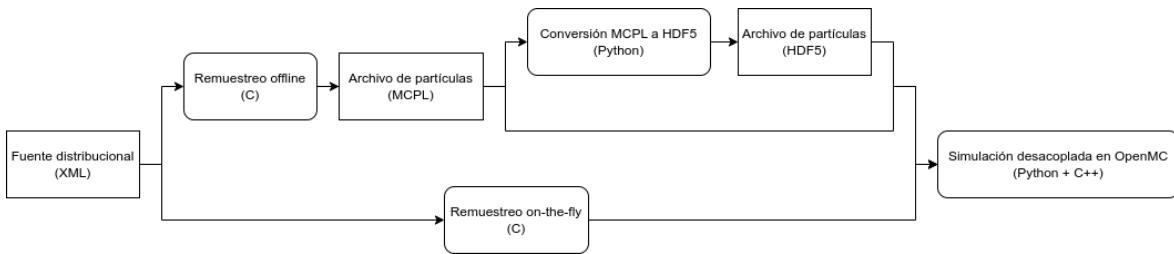


Figura 2.4: Etapa de producción: uso del archivo XML para generar nuevas partículas de forma offline o on-the-fly. Bloques con esquinas redondeadas representan procesos computacionales, mientras que los bloques rectangulares indican archivos que se registran o leen desde disco.

Capítulo 3

Metodología para la generación de fuentes Monte Carlo mediante histogramas multidimensionales

3.1. Introducción general al método

La metodología propuesta se basa en el procesamiento de archivos de partículas generados en simulaciones Monte Carlo previas, específicamente usando OpenMC, para definir fuentes distribucionales para simulaciones subsiguientes.

3.2. Definición del espacio de fases ($E-r-\Omega$)

Para aproximar correctamente las distribuciones y correlaciones del espacio de fases se consideraron seis variables para representar la energía, posición y dirección: tres coordenadas espaciales (x, y, z), dos variables direccionales definidas en coordenadas esféricas ($\mu = \cos(\theta), \phi$) y una variable energética (E o letargía $\ln(E_0/E)$). En la Figura 3.1 se observa una representación de las variables del espacio de fases. En situaciones como la considerada en este trabajo, donde la fuente se registra sobre una superficie plana, es posible rotar el sistema de coordenadas de modo que dicha superficie resulte perpendicular al eje z . Por lo tanto la coordenada z permanece constante, permitiendo representar el espacio de fases con cinco variables.

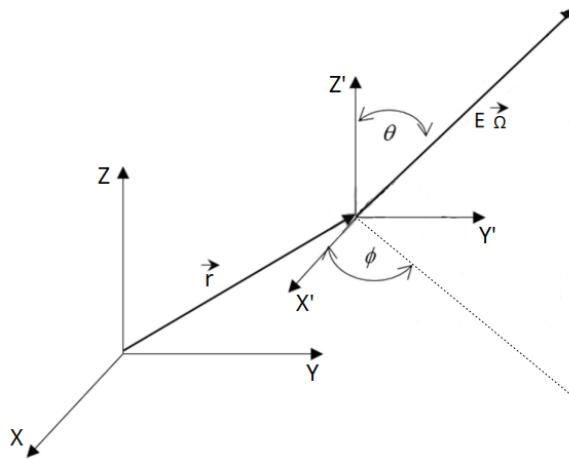


Figura 3.1: Representación del espacio de fases con las variables $(E, x, y, z, \theta, \phi)$, donde E es la energía, x , y y z son coordenadas espaciales y θ y ϕ son las variables direccionalles.

3.3. Procesamiento del archivo de partículas original

3.3.1. Preprocesamiento del archivo de partículas

Inicialmente, las partículas provenientes de la simulación Monte Carlo original se filtran seleccionando únicamente aquellas que se propagan hacia la región de interés y se separan las variables relevantes mencionadas, además del peso estadístico (*weight*). Esta selección se realiza porque, en la siguiente etapa de simulación, la geometría ya no incluye la región anterior a la superficie de registro —es decir, la zona desde donde provienen originalmente las partículas—, y se asume un vacío en su lugar. Por lo tanto, sólo tiene sentido conservar aquellas partículas que se propagan hacia la región de interés. Las partículas que se dirigen en sentido opuesto no son relevantes para la simulación posterior, ya que en caso de que alguna partícula fuese a regresar por retrodispersión, dicho comportamiento ya está estadísticamente incorporado en la lista original de partículas, producto de la simulación completa previa. En la Tabla 3.1 se presenta un ejemplo de la estructura típica de un archivo de partículas.

3.3.2. Utilización de histogramas macro y micro

La metodología desarrollada en este trabajo para aproximar distribuciones y correlaciones multidimensionales en el espacio de fases de partículas provenientes de simulaciones Monte Carlo se basa en un esquema combinado de histogramas macro y micro. Esta aproximación permite gestionar eficientemente grandes volúmenes de datos manteniendo una representación precisa de la información esencial del conjunto original de partículas.

Los histogramas macro son subdivisiones jerárquicas del espacio de fases, realiza-

Partícula N°	Letargía	x [cm]	y [cm]	μ	ϕ [rad]	Peso estadístico
1	4.15	0.23	-1.10	0.85	3.14	1.00
2	4.95	-0.75	0.40	0.65	1.57	0.57
3	5.05	1.10	0.70	0.45	0.78	1.00
4	5.30	-0.50	-0.90	0.60	2.35	0.33
5	4.85	0.85	-0.20	0.95	1.25	1.00
:	:	:	:	:	:	:
100000	5.00	0.10	0.55	0.70	0.95	0.99

Tabla 3.1: Ejemplo ilustrativo de la estructura típica de un archivo de partículas. El archivo original suele contener cientos de miles de partículas.

das siguiendo un orden determinado por quien utiliza la herramienta. Por ejemplo, un posible orden es la letargía, seguida por las coordenadas espaciales (X , Y), y posteriormente por la dirección (μ , ϕ). Este procedimiento inicia dividiendo el conjunto original de partículas en macrogrupos según la primera variable elegida (por ejemplo, letargía). Cada macrogrupo así generado es posteriormente subdividido en nuevos macrogrupos en función de la siguiente variable (por ejemplo, la coordenada X), repitiéndose este proceso de manera iterativa para cada variable subsiguiente. El resultado es una estructura jerárquica en forma de árbol, donde cada rama representa un subconjunto específico del archivo original de partículas, ya que incluye las cinco variables consideradas. Cada nodo en la rama corresponde a una variable dentro de dicho subconjunto. En la Figura 3.2 se ilustra este procedimiento de subdivisión jerárquica del espacio de fases. Este método implementado presenta diferencias respecto al esquema empleado en trabajos anteriores [3], donde los macrogrupos y microgrupos eran construidos de forma cíclica, es decir, la discretización de cada variable consideraba simultáneamente la influencia de las demás. En contraste, en la implementación aquí desarrollada, los macrogrupos y microgrupos de la primera variable seleccionada se construyen sin referencia a las restantes, y a partir de esta primera discretización se procede con la segmentación de la siguiente variable, considerando únicamente las divisiones ya establecidas. De este modo, el orden en que se procesan las variables define una jerarquía en la cual cada nivel se construye condicionado por las particiones realizadas en los niveles anteriores, y no en función de variables aún no discretizadas.

Este esquema tiene como objetivo capturar las correlaciones entre las variables del espacio de fases. Cada nodo o macrogrupo resultante comparte similitudes en todas las variables consideradas hasta ese nivel jerárquico. Por ejemplo, un macrogrupo particular tendrá partículas con distribuciones similares en letargía, coordenadas espaciales (X , Y), y dirección (μ), lo cual garantiza que la distribución restante en la última variable (ϕ) refleje adecuadamente las correlaciones subyacentes en los datos.

Cabe destacar que, en la etapa de macrodiscretización, se utilizan histogramas con

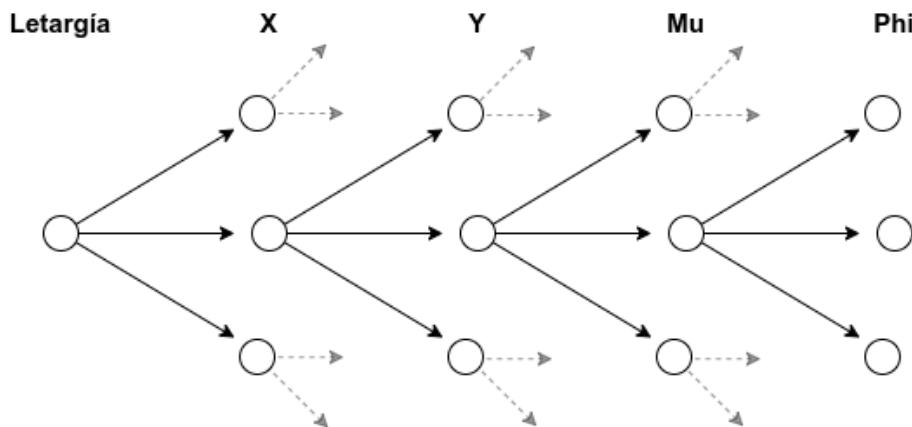


Figura 3.2: Esquema ilustrativo de la estructura jerárquica de macrogrupos y microgrupos formando un árbol. Se resalta la jerarquía entre variables.

baja resolución debido al crecimiento exponencial en la cantidad total de grupos generados, dado que el número total de macrogrupos es producto del número de divisiones en cada variable. Un número excesivamente alto de divisiones podría conducir a subconjuntos con muy pocas partículas, comprometiendo la calidad estadística.

Una vez establecidos los macrogrupos, cada nodo es discretizado con histogramas micro, los cuales poseen una mayor resolución para representar detalladamente la distribución de cada variable dentro del subconjunto. Esta discretización se realiza ponderando por el peso estadístico, lo que permite conservar la importancia relativa de cada evento. A partir de los histogramas micro así construidos, se calcula la función de frecuencia acumulada, la cual es posteriormente normalizada para definir una distribución de probabilidad que pueda ser utilizada en el proceso de remuestreo. Sin embargo, es crucial controlar esta resolución para evitar reproducir el ruido estadístico presente en los datos originales, especialmente cuando se cuenta con un número reducido de partículas en el subconjunto.

La combinación de histogramas macro y micro permite obtener una aproximación eficiente y precisa de la distribución multidimensional del conjunto original, conservando las correlaciones entre las variables del espacio de fases. Esta estructura de histogramas multidimensionales representa, en síntesis, la información esencial extraída del archivo original de partículas para ser utilizada en simulaciones posteriores.

3.3.3. Métodos de discretización aplicados a histogramas macro y micro

En este trabajo se desarrollaron dos métodos de discretización (*bineado*) aplicables tanto a los histogramas macro como a los micro. El primero es el *bineado uniforme*, que consiste en dividir la distribución de la variable en estudio utilizando bins equiespaciados. Este método tiene la ventaja de ser sencillo y rápido en su implementación,

pero presenta la desventaja de no poder capturar adecuadamente cambios abruptos o picos aislados en las distribuciones.

El segundo método, denominado *bineado* adaptativo, consiste en realizar inicialmente un *bineado* uniforme con una cantidad moderada de bins especificada por quien utiliza la herramienta. Luego, este *bineado* inicial se refina iterativamente agregando nuevos bins donde la distribución estimada difiere en mayor medida respecto a la distribución original. El criterio para añadir un nuevo bin se basa en la comparación de la función de distribución acumulada (FDC) del *bineado* actual con la FDC de la distribución original calculada con alta resolución. En cada iteración, se identifica el punto con la mayor diferencia absoluta entre ambas FDC, donde se coloca un borde de bin adicional. En el [Apéndice A](#) se presenta un ejemplo de la implementación de este método.

Este enfoque adaptativo permite asignar mayor resolución donde existe una mayor cantidad de peso estadístico, y menor resolución donde el peso es escaso, logrando así un mejor seguimiento de las zonas críticas y un adecuado suavizado en regiones con poca estadística.

Además, debido a que la cantidad de partículas generalmente disminuye al profundizar en las ramas del árbol jerárquico, puede ser necesario utilizar una resolución decreciente en las discretizaciones, tanto para los histogramas macro como para los micro, lo cual también ha sido implementado en este trabajo.

Finalmente, en aquellos casos en que quien utiliza el código conozca previamente la existencia de cambios abruptos o picos específicos en la distribución, es posible informar manualmente al método sobre la ubicación de estos fenómenos, estableciendo bordes de bin específicamente en esos puntos para mejorar la precisión de la discretización.

3.4. Remuestreo de partículas en simulaciones Monte Carlo subsecuentes

El proceso de generación de partículas a partir de la información guardada en los histogramas multidimensionales implica:

1. Generar un número pseudoaleatorio entre 0 y 1.
2. Interpolar dicho número en la distribución acumulada normalizada de la variable raíz del árbol.
3. Avanzar secuencialmente por las ramas del árbol determinando valores de las variables subsecuentes hasta obtener un conjunto completo de variables del espacio de fases.

En la Figura 3.3 se ejemplifica el proceso de generación de partículas descrito previamente. En el caso particular de trabajar con cinco variables, se repetirá la generación de números pseudoaleatorios cinco veces consecutivas, obteniéndose así los cinco valores correspondientes de las variables. Cabe destacar que durante el remuestreo de la quinta variable no se generará un macrogrupo adicional, dado que esta es la última variable a muestrear y no existen más niveles en el árbol.

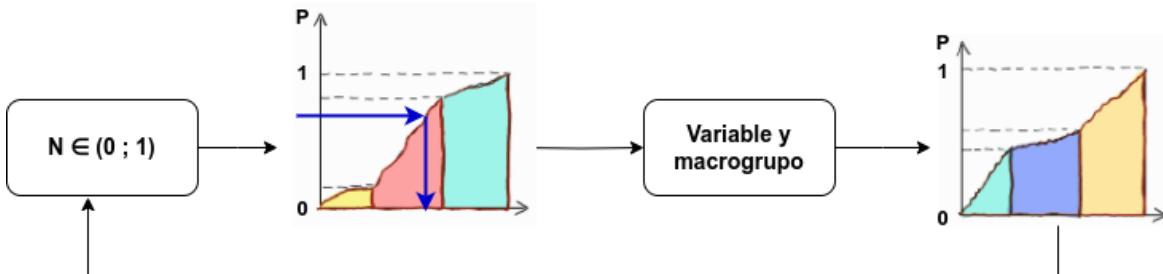


Figura 3.3: Esquema ilustrativo del proceso secuencial de generación de partículas a partir de la distribución jerárquica guardada, resaltando el muestreo sucesivo mediante números pseudoaleatorios.

A través de este proceso es posible remuestrear nuevas partículas para la siguiente etapa de la simulación. Existen dos formas de integrar esta funcionalidad con OpenMC:

- Una opción consiste en realizar, de forma *offline*, el muestreo de una cantidad predeterminada de partículas y guardar sus propiedades en un archivo de partículas. Este archivo puede ser luego utilizado por OpenMC mediante su opción de simulación desde el modo de fuente `FileSource` implementado en el programa.
- Alternativamente, se puede ejecutar OpenMC con una fuente `HistogramSource`, definida *ad hoc* y configurada mediante un archivo XML. En este caso, OpenMC accede directamente al árbol de histogramas durante la simulación y genera cada partícula *on-the-fly*, lo que reduce significativamente el uso de memoria y evita la necesidad de almacenar archivos de partículas intermedios voluminosos.

Esta última funcionalidad fue incorporada en el contexto del presente trabajo, mediante la definición de una nueva clase de fuente en el código fuente de OpenMC en C++ [17] [18]. Se desarrollaron las estructuras necesarias en C para efectuar el muestreo desde histogramas multidimensionales, y se integraron con la API de OpenMC en Python, permitiendo así que los histogramas puedan ser cargados y utilizados dentro del flujo habitual de simulación. El objetivo principal de esta implementación es facilitar el uso práctico de la herramienta.

3.5. Implementación computacional

La implementación computacional de la metodología descrita en este capítulo se encuentra desarrollada y documentada en el **Apéndice B**. En particular, se presentan los códigos elaborados en **Python**, **C** y **C++** que posibilitan la generación y manejo de los histogramas multidimensionales generados a partir del archivo original de partículas. Dichos histogramas se configuran mediante parámetros específicos definidos por quien utiliza la herramienta, tales como número y tipo de bines, así como el orden en que se procesan las variables del espacio de fases.

Asimismo, se documenta en detalle la integración realizada con **OpenMC**, incluyendo las modificaciones llevadas a cabo en su *API* de **Python** y en su código fuente en **C++**, para permitir la generación *on-the-fly* de partículas durante las simulaciones Monte Carlo subsecuentes.

Capítulo 4

Validación del método implementado: canal de vacío rodeado por agua

En el presente capítulo se verifica la implementación del método desarrollado para la generación de fuentes Monte Carlo mediante histogramas multidimensionales. El caso de estudio seleccionado consiste en un canal de vacío rodeado por agua liviana. El modelo implementado se detallará en la siguiente sección. El objetivo principal de este capítulo es analizar el desempeño del método propuesto en términos de conservación del flujo escalar a lo largo del modelo, así como evaluar la preservación del espectro energético y la corriente neutrónica en diferentes regiones del canal. Se ha seleccionado este modelo debido a que es un problema sensible ante perturbaciones de la fuente, lo que permite evaluar la eficacia del método de remuestreo implementado.

Con el objetivo de verificar el método de remuestreo desarrollado, se realizó en primer lugar una simulación desde la entrada del modelo hasta una superficie de registro ubicada en una posición intermedia de la geometría. Esta simulación tuvo como único propósito generar un archivo de partículas, el cual será procesado para construir una fuente distribucional.

Posteriormente, se llevó a cabo una segunda simulación, también iniciada desde la entrada del sistema, pero extendida hasta una superficie ubicada más lejos de la fuente. A diferencia de la anterior, esta simulación tuvo por objetivo obtener resultados de referencia para uso posterior: en particular, el perfil convergido del flujo escalar a lo largo del canal, así como la corriente y el espectro neutrónico incidente en la superficie más alejada de la fuente. Estos resultados de referencia son utilizados en etapas posteriores del trabajo como base para la validación del método de remuestreo propuesto.

Finalmente, se procesó el archivo de partículas grabado en la superficie intermedia

de la primera simulación para construir una fuente distribucional basada en histogramas multidimensionales. Esta fuente se utilizó en una tercera simulación iniciada directamente desde la posición intermedia, obteniendo así una estadística suficientemente convergida hasta la superficie más alejada de la fuente. Esta última simulación permitió realizar una comparación directa con los resultados obtenidos en la segunda simulación, evaluando así el método de generación de fuentes implementado, tal como se resume esquemáticamente en la Figura 4.1.

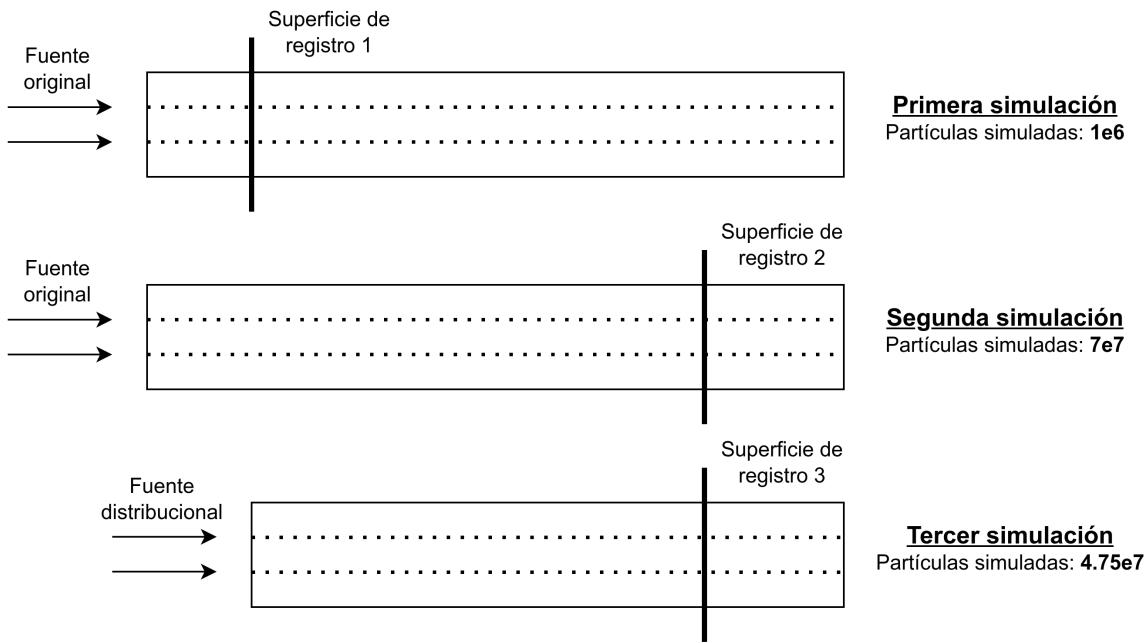


Figura 4.1: Esquema del proceso de remuestreo y validación del método implementado.

4.1. Descripción del modelo simulado en OpenMC

El modelo utilizado en las simulaciones consiste en un canal interno de vacío rodeado lateralmente por agua liviana. El sistema se diseñó como un paralelepípedo de agua con dimensiones $15\text{ cm} \times 15\text{ cm} \times 100\text{ cm}$, que alberga en su interior un canal de vacío con sección transversal cuadrada de $3\text{ cm} \times 3\text{ cm}$, orientado en dirección del eje z . En la Figura 4.2 se presenta un esquema del modelo implementado en OpenMC.

La fuente de neutrones utilizada se definió sobre la cara de entrada del sistema (ubicada en $z = 0\text{ cm}$) y consistió en neutrones monoenergéticos con energía de $E = 1\text{ MeV}$ y colimados en dirección del eje z (con coseno direccional $\mu = \cos(\theta) = 1$). Esta fuente se extiende en toda la superficie transversal al tubo, tanto en la región de agua como en la región de vacío. Esta configuración genera dos poblaciones de neutrones claramente diferenciadas: la primera, formada por neutrones que atraviesan el canal de vacío sin colisiones y mantienen tanto su energía inicial como su dirección;

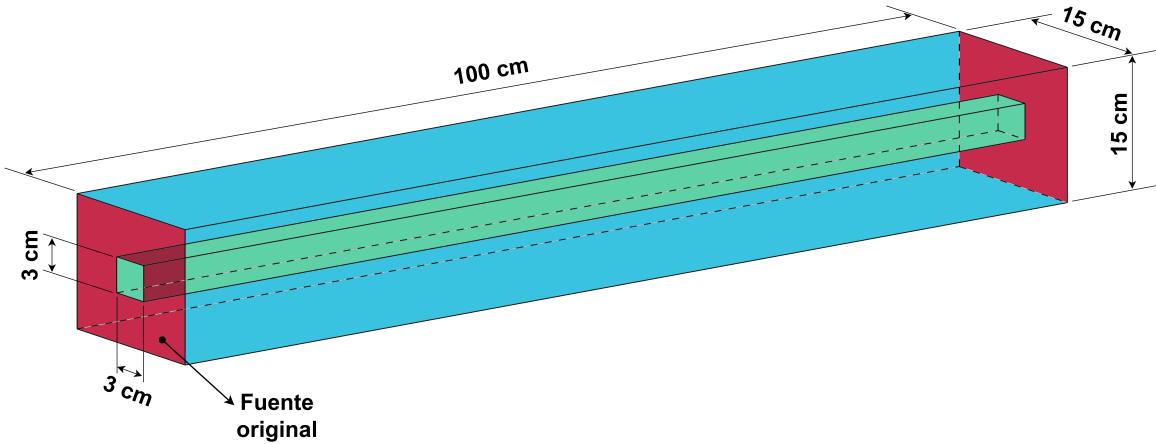


Figura 4.2: Esquema del modelo simulado en OpenMC.

y la segunda, constituida por neutrones que interactúan con el moderador de agua, sufriendo dispersión angular y pérdida energética.

Para el análisis y posterior generación de la fuente, se registró un archivo de partículas sobre una superficie intermedia situada en $z = 15$ cm. Este archivo contiene información sobre ambas poblaciones mencionadas, es decir, neutrones no colisionados y neutrones dispersados.

Durante las simulaciones se determinó el flujo escalar neutrónico a lo largo del canal, tanto dentro del canal de vacío como en el moderador circundante.

Para optimizar la estadística en regiones con baja presencia de partículas, particularmente en el agua, se implementó la técnica de ventanas de peso disponible en OpenMC. Este método actualiza los pesos de las partículas durante la simulación, permitiendo obtener mejorar estadística.

4.2. Descripción del archivo de partículas original

El archivo de partículas fue registrado en una primera simulación del tubo de vacío desde el comienzo de la geometría. Corresponde a partículas registradas en una superficie ubicada a 15 cm del inicio de la geometría. La lista contiene tanto neutrones que atraviesan la sección del canal, como neutrones que se transportan por el agua. Sin embargo, la mayor proporción de peso estadístico se encuentra dentro del tubo de vacío, lo cual se corrobora tanto en la proyección $X-Y$, donde se observa una distribución más intensa contenida en un cuadrado correspondiente a la sección del tubo, como en la Tabla 4.1, que presenta la cantidad de partículas y peso estadístico registrado por región. La Figura 4.3 muestra la proyección de los neutrones en el plano $X-Y$, donde

$$\eta(x, y) = \frac{1}{\Delta x \Delta y} \int_0^\infty du \int_{\Delta x} dx \int_{\Delta y} dy \int_{2\pi^+} d\Omega (\boldsymbol{\Omega} \cdot \mathbf{n}) \varphi(\mathbf{r}, u, \boldsymbol{\Omega}) \quad (4.1)$$

representa la corriente de neutrones en $\frac{n}{s \text{ cm}^2}$ que atraviesa la superficie definida por el plano $X-Y$, en función de las coordenadas x e y . En esta variable η se integra en todo el dominio de las variables que no son parámetro, para dejar la intensidad de neutrones en función únicamente de las variables parámetros con las cuales se discretiza la lista de partículas. A su vez, en esta notación Δ se utiliza para notar el ancho de un bin de una variable. En todos los casos, se informa el valor normalizado por neutrón de fuente, denotado como S_p .

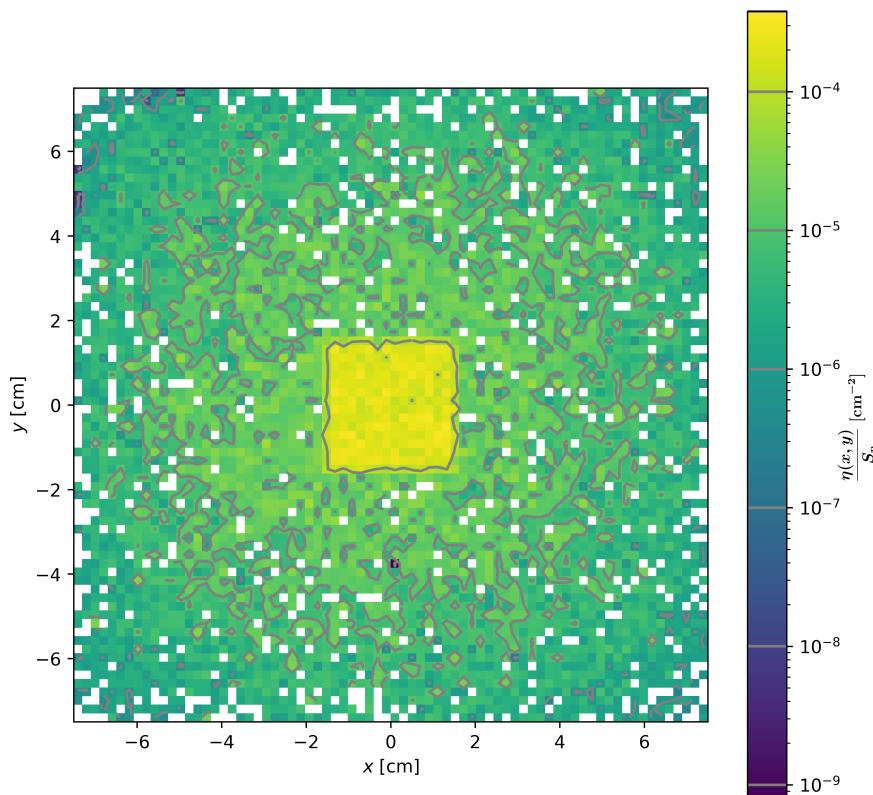


Figura 4.3: Distribución de X vs Y para el archivo de partículas registrado en la primera simulación del tubo de vacío.

Este conjunto contiene dos poblaciones de neutrones claramente diferenciadas. La primera corresponde a neutrones que no han sufrido colisiones: todos ellos presentan dirección $\mu = 1$ y una letargía fija correspondiente a una energía de $E = 1$ MeV, donde letargía $u = \ln(E_0/E)$ con $E_0 = 20$ MeV. Estos valores coinciden con la configuración de la fuente original, la cual fue definida como perfectamente colimada y monoenergética. La segunda población está compuesta por neutrones que han interactuado con el medio: en este caso, las distribuciones de μ y letargía se extienden de forma continua, reflejando los efectos introducidos por las colisiones. Este comportamiento se observa en la Figura 4.4, donde se presentan las distribuciones de letargía y μ correspondientes al archivo de partículas registrado en la primera simulación del tubo de vacío. En dichas

figuras,

$$\eta(u) = \frac{1}{\Delta u} \int_{\Delta u} du \int_A dA \int_{2\pi^+} d\Omega (\boldsymbol{\Omega} \cdot \mathbf{n}) \varphi(\mathbf{r}, u, \boldsymbol{\Omega}) \quad (4.2)$$

representa la corriente de neutrones en $\frac{n}{s}$ que atraviesa la superficie en función de la letargía u y

$$\eta(\mu) = \frac{1}{\Delta \mu} \int_0^\infty du \int_A dA \int_{2\pi} d\phi \int_{\Delta \mu} d\mu (\boldsymbol{\Omega} \cdot \mathbf{n}) \varphi(\mathbf{r}, u, \boldsymbol{\Omega}) \quad (4.3)$$

representa la corriente de neutrones en $\frac{n}{s}$ que atraviesa la superficie en función del coseno direccional μ .

Región	Área [cm ²]	Cantidad registrada [#]		Densidad [#/cm ²]	
		Partículas	Peso estad.	Partículas	Peso estad.
Vacio	9	62.523	46.862	6.947	5.207
Agua	216	390.412	44.938	1.807	208
Total	225	452.935	91.800	—	—

Tabla 4.1: Resumen de partículas y peso estadístico registrados en la superficie intermedia ($z = 15$ cm) para cada región. Se informa también el área de cada región, así como la densidad de partículas y peso estadístico.

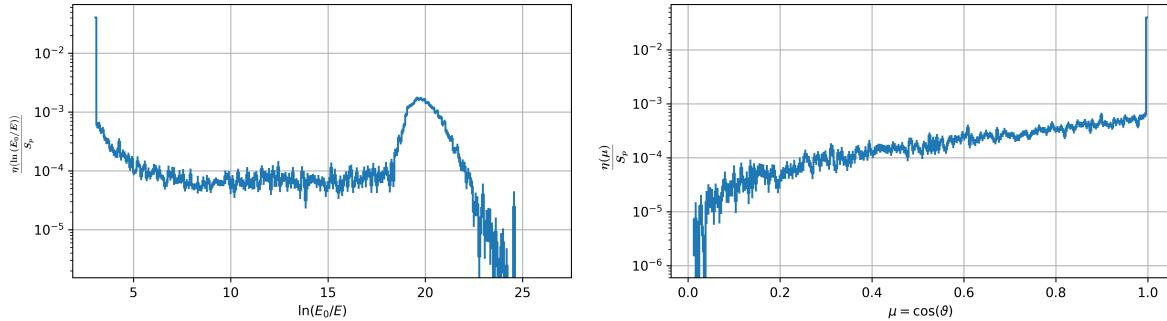


Figura 4.4: Distribuciones de letargía y μ para el archivo de partículas registrado en la primera simulación del tubo de vacío, con escala logarítmica en el eje y. Se observa la presencia de dos poblaciones: una concentrada en $\mu = 1$ y letargía fija correspondiente a una energía de $E = 1$ MeV, montada sobre una distribución continua de neutrones colisionados.

Esta doble estructura se evidencia particularmente en los gráficos bidimensionales de letargía vs. μ , donde ambas poblaciones aparecen como conjuntos diferenciados. Tal como se muestra en la Figura 4.5, la corriente de neutrones no colisionados se concentra en un único punto en $\mu = 1$ y letargía fija correspondiente a una energía de $E = 1$ MeV, mientras que los neutrones colisionados forman una distribución extendida continua, con un agrupamiento en la región de termalización. En dicha figura se presenta la

distribución de corriente en $\frac{n}{s}$ en función de μ y u , definida como

$$\eta(\mu, u) = \frac{1}{\Delta\mu\Delta u} \int_{\Delta u} du \int_A dA \int_{2\pi} d\phi \int_{\Delta\mu} d\mu (\boldsymbol{\Omega} \cdot \mathbf{n}) \varphi(\mathbf{r}, u, \boldsymbol{\Omega}). \quad (4.4)$$

Este fenómeno plantea un desafío para los métodos de muestreo, al requerir una representación precisa tanto de distribuciones tipo delta como de distribuciones extendidas.

Tanto en la Figura 4.3 como en la Figura 4.5 puede observarse que existen regiones del espacio de fases donde no se registran neutrones. Esto se debe a que el archivo de partículas original contiene una cantidad finita de neutrones, lo que limita la cobertura en determinadas regiones del espacio de fases. El objetivo del método propuesto consiste en remuestrear dicho archivo original para generar un nuevo conjunto de partículas que preserve las correlaciones presentes, pero que a su vez mejore la estadística en aquellas regiones originalmente poco representadas.

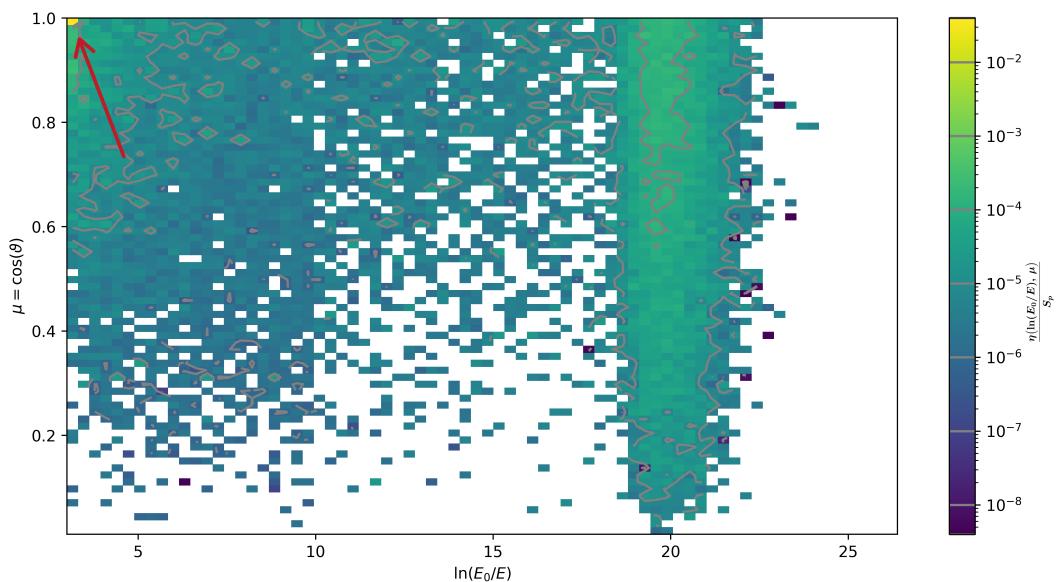


Figura 4.5: Distribuciones de letargía vs μ para el archivo de partículas original. Se observa la presencia de un conjunto de neutrones graficados como un píxel en $\mu = 1$ y letargía correspondiente a $E = 1$ MeV. A su vez se observa la presencia de un segundo conjunto de neutrones colisionados, que se distribuyen en el plano de forma continua, con un agrupamiento sobre letar-gía = 20, indicando la termalización de los neutrones.

4.3. Procesamiento preliminar del archivo de partículas

4.3.1. Metodología del análisis

El análisis realizado en este capítulo se centra en evaluar la influencia del tipo de *bineado* aplicado, así como la incorporación de bordes manuales en la segmentación

del espacio de fases. Estos bordes manuales se definen en la configuración del método y permiten segregar en macro grupos las variables especificadas. Los tipos de *bineado* considerados son:

- Caso 1: *bineado* uniforme sin bordes manuales.
- Caso 2: *bineado* uniforme con bordes definidos por quien utiliza el programa.
- Caso 3: *bineado* adaptativo sin bordes manuales.
- Caso 4: *bineado* adaptativo con bordes definidos por quien utiliza el programa.

Los bordes manuales son establecidos debido al conocimiento previo de la fuente utilizada para diferenciar poblaciones de neutrones. Concretamente, se incorporan bordes en la variable letargía y en la dirección cosenoidal (μ) para distinguir los neutrones que no han colisionado. Adicionalmente, se incluyen bordes espaciales en las coordenadas X e Y para separar los neutrones que atraviesan el tubo de vacío respecto de aquellos que interactúan con el agua circundante.

Con el objetivo de realizar un análisis consistente y evitar sesgos derivados del procedimiento, se mantiene constante en todos los casos el orden de procesamiento de las variables y la configuración de los histogramas. Esta etapa preliminar permite una comparación entre los cuatro métodos de *bineado* evaluados en esta sección. El orden seleccionado para el procesamiento es [`letargía, X, Y, μ, φ`]. Esta elección responde a que al inicio del procesamiento, la discretización se aplica sobre el conjunto completo de partículas, mientras que las variables procesadas posteriormente actúan sobre subconjuntos cada vez más reducidos. Por este motivo, se eligió comenzar con la letargía para priorizar describir el comportamiento espectral. A continuación se aplican las variables espaciales (X e Y), con el fin de preservar correctamente la ubicación de las partículas en regiones geométricamente diferenciadas (vacío o agua). Finalmente, se dejó para el último lugar la variable ϕ . Si bien dentro del agua y en el vacío se espera una distribución aproximadamente uniforme en ϕ , esta isotropía puede alterarse en las cercanías de las interfaces agua–vacío. En esas zonas coexisten neutrones que escapan desde el agua al vacío, generando distribuciones no uniformes en ϕ . Sin embargo, dado que la mayor parte de las partículas se encuentra en regiones donde la distribución es aproximadamente uniforme, se considera que ϕ es la variable con menor variabilidad, y por ello se la procesa en último lugar.

Asimismo, se conservan fijos los números de bins utilizados en los histogramas macro y micro: se emplean [5, 5, 5, 5] bins para los macrogrupos y [36, 36, 36, 36] para los microgrupos, respectivamente. Esta configuración constante garantiza que las comparaciones entre métodos se basen únicamente en el esquema de discretización utilizado y no en diferencias en la resolución.

Posteriormente, el análisis se estructura en tres partes:

- **Distribuciones de letargía:** se presentan y comparan las distribuciones de letargía obtenidas para cada uno de los cuatro esquemas evaluados, comparándolas contra la distribución registrada en el archivo original.
- **Distribuciones de μ :** se analizan de forma análoga las distribuciones de la variable direccional $\mu = \cos \theta$, evaluando en qué medida se preserva la estructura angular del haz en cada uno de los casos.
- **Distribuciones espaciales $X-Y$:** se analizan los resultados mediante mapas bidimensionales para cada una de las configuraciones, comparando las distribuciones generadas contra las distribuciones originales.

4.3.2. Distribuciones de letargía

A continuación se analizan las distribuciones remuestreadas de letargía obtenidas mediante los 4 casos de configuración analizados.

En la Figura 4.6, correspondiente al caso 1 de *bineado* uniforme sin bordes, se observa que esta configuración presenta limitaciones en regiones de alta densidad estadística. En particular, la delta en la zona de letargía correspondiente a una energía de $E = 1$ MeV —asociada a neutrones no colisionados— es reemplazada por un escalón con el ancho del bin, perdiéndose el comportamiento tipo delta. Asimismo, en la región de termalización, la resolución es insuficiente para capturar adecuadamente los cambios de forma.

La Figura 4.7 muestra el caso 2 en el que se introduce un borde manual para separar la letargía correspondiente a una energía de $E = 1$ MeV. Esta intervención permite mejorar la representación de la delta, capturando con mayor fidelidad los neutrones no colisionados. Sin embargo, el resto de la distribución sigue estando limitada por la resolución uniforme, en particular en la zona de termalización.

En la Figura 4.8, se presenta el resultado del caso 3 utilizando histogramas adaptativos sin bordes manuales. Este método asigna una mayor densidad de bins en zonas con mayor estadística, lo que permite una mejor representación tanto de la delta en letargía correspondiente a una energía de $E = 1$ MeV como de la región de termalización. Se observa un seguimiento más suave y detallado de la distribución, y una reducción del error relativo en las zonas relevantes.

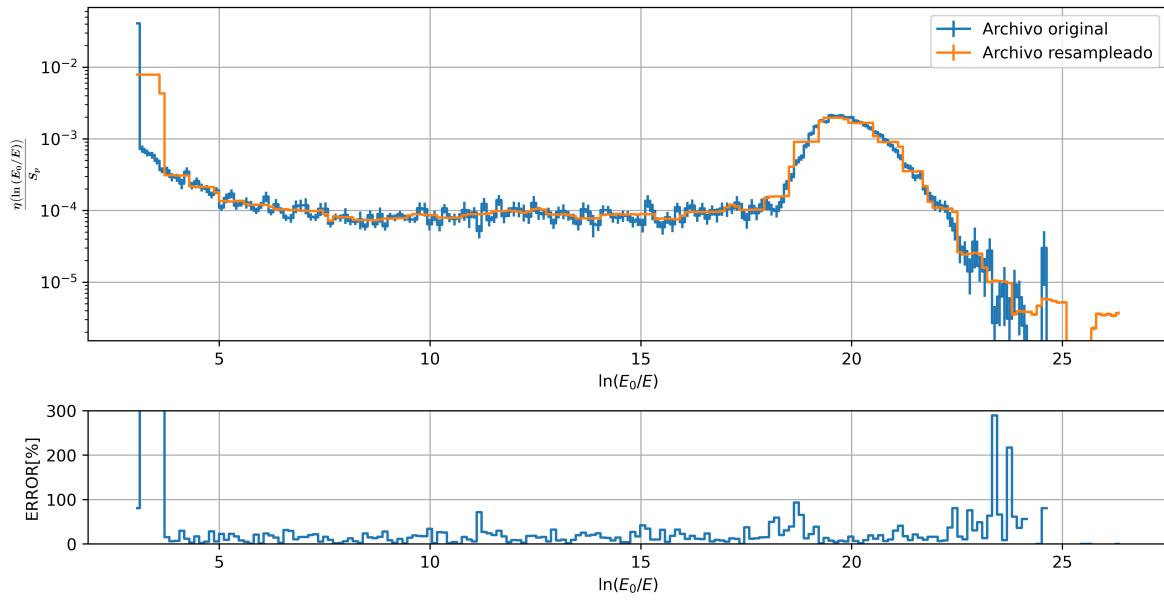


Figura 4.6: Distribución remuestreada de letargía para el caso de *bineado* uniforme sin bordes manuales.

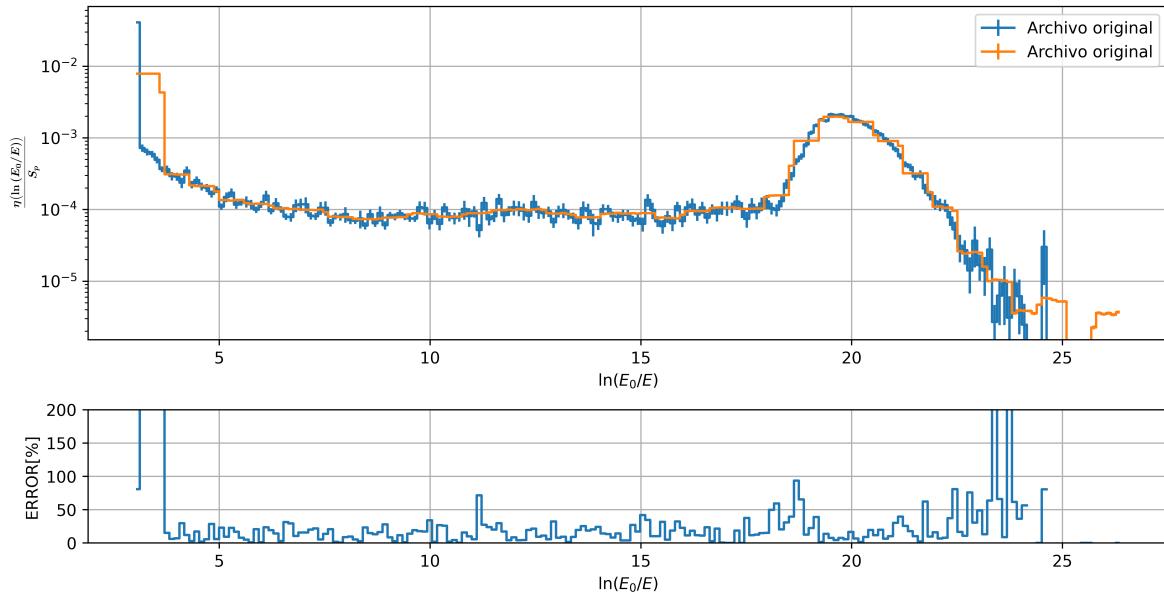


Figura 4.7: Distribución remuestreada de letargía para el caso de *bineado* uniforme con borde manual en la zona de letargía correspondiente a una energía de $E = 1$ MeV.

Finalmente, en la Figura 4.9, se muestra el resultado del caso 4 con el método adaptativo con bordes manuales. No se observan diferencias significativas con respecto al caso sin bordes, lo que demuestra que el algoritmo es capaz de identificar y segmentar eficientemente las zonas relevantes de forma autónoma, incluso sin información previa de la fuente brindada por el borde manual.

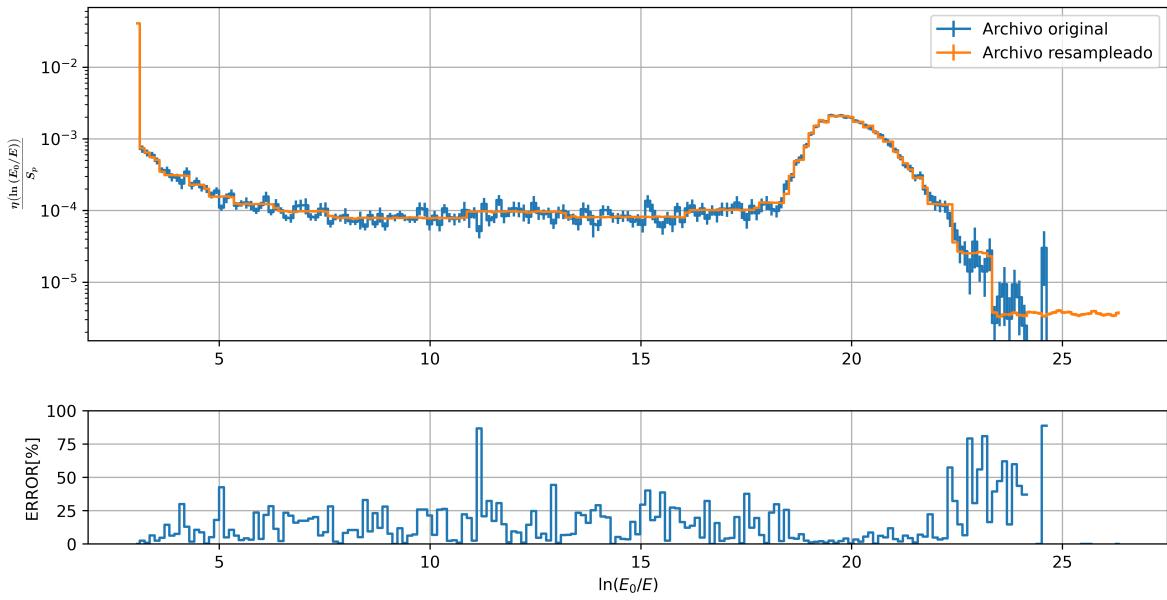


Figura 4.8: Distribución remuestreada de letargía para el caso de *bineado* adaptativo sin bordes manuales.

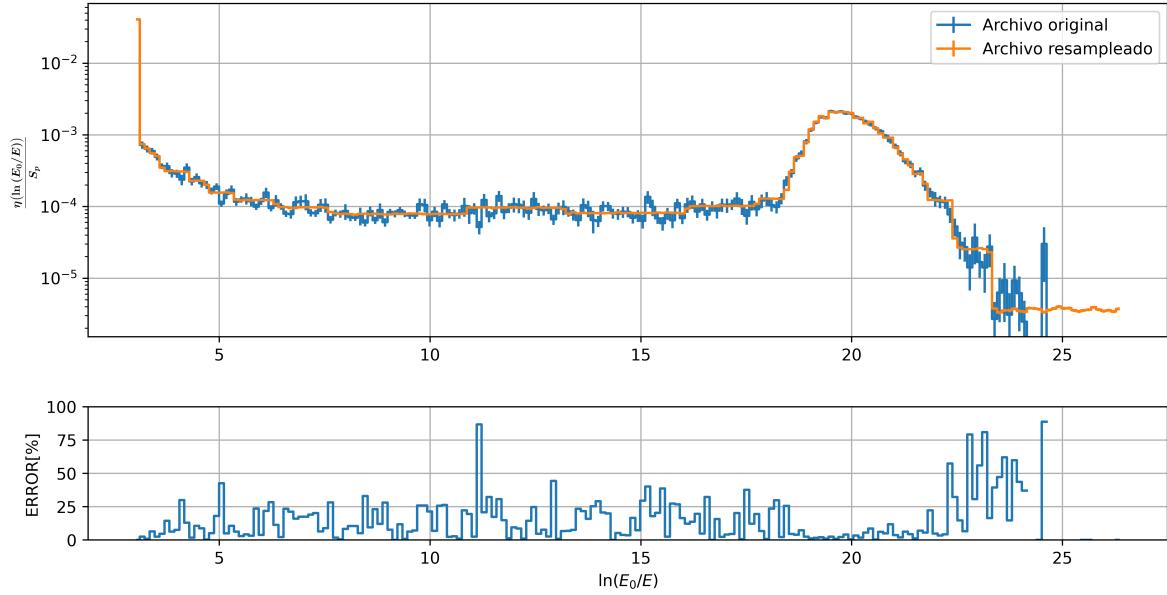


Figura 4.9: Distribución remuestreada de letargía para el caso de *bineado* adaptativo con borde manual en la zona de letargía correspondiente a una energía de $E = 1$ MeV.

4.3.3. Distribuciones de μ

Se analizan a continuación las distribuciones remuestreadas de la variable direccional $\mu = \cos(\theta)$, obtenidas mediante las cuatro configuraciones analizadas. Se evalúa la capacidad de cada método para representar adecuadamente la delta asociada a partículas colimadas ($\mu = 1$) y el comportamiento general de la distribución.

En la Figura 4.10, correspondiente al caso 1 de *bineado* uniforme sin bordes ma-

niales, se observa que la delta en $\mu = 1$ es reemplazada por un escalón, debido a que el bin abarca toda la región colimada. Este efecto deteriora significativamente la representación en esa zona. Sin embargo, el resto de la distribución es tratada de forma aceptable, suavizando la distribución.

En la Figura 4.11, se introduce un borde manual para formar un bin de ancho 10^{-9} en la zona de $\mu = 1$, lo que permite aislar correctamente la región colimada. Como resultado, se recupera la forma tipo delta en esa zona, y se mantiene una buena representación en el resto de la distribución.

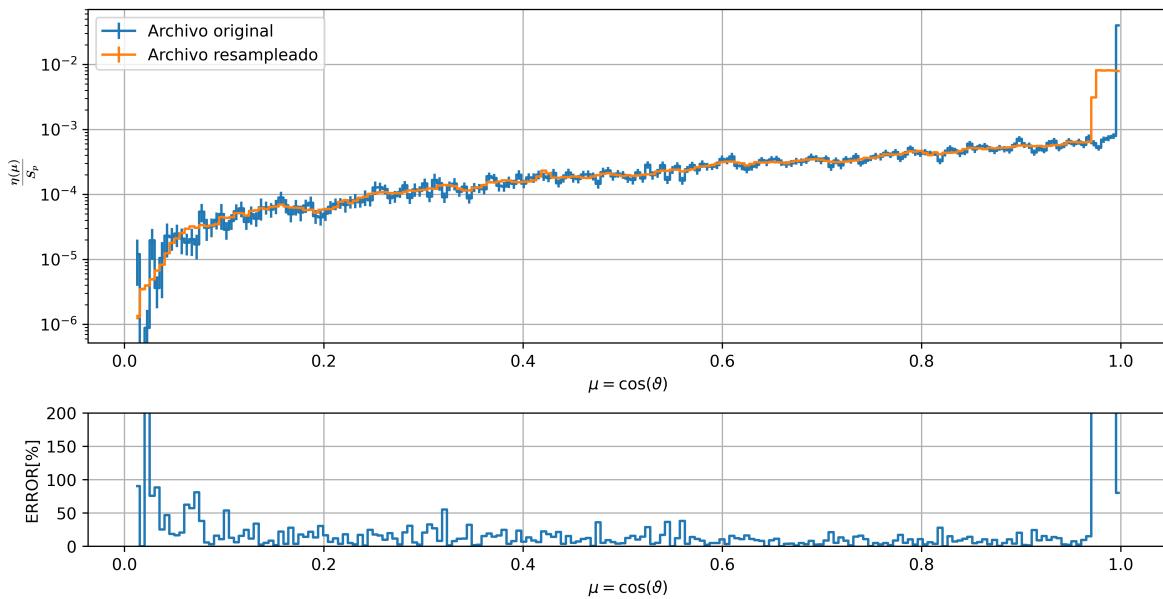


Figura 4.10: Distribución remuestreada de μ para el caso 1 de *bineado* uniforme sin bordes manuales.

La Figura 4.12 muestra los resultados del caso 3 obtenidos con histogramas adaptativos sin bordes manuales. En este caso, el método logra seguir de manera precisa la forma original, tanto en la delta como en el resto de la distribución. Sin embargo, este seguimiento resulta excesivo, reproduciendo el ruido estadístico del archivo de partículas original, lo cual no es deseable. Este efecto se debe a la elevada cantidad de bins asignados: como μ es la cuarta variable en el orden de segmentación, se trabaja sobre subconjuntos reducidos de partículas, lo cual hace que 36 bins resulte una resolución desproporcionada.

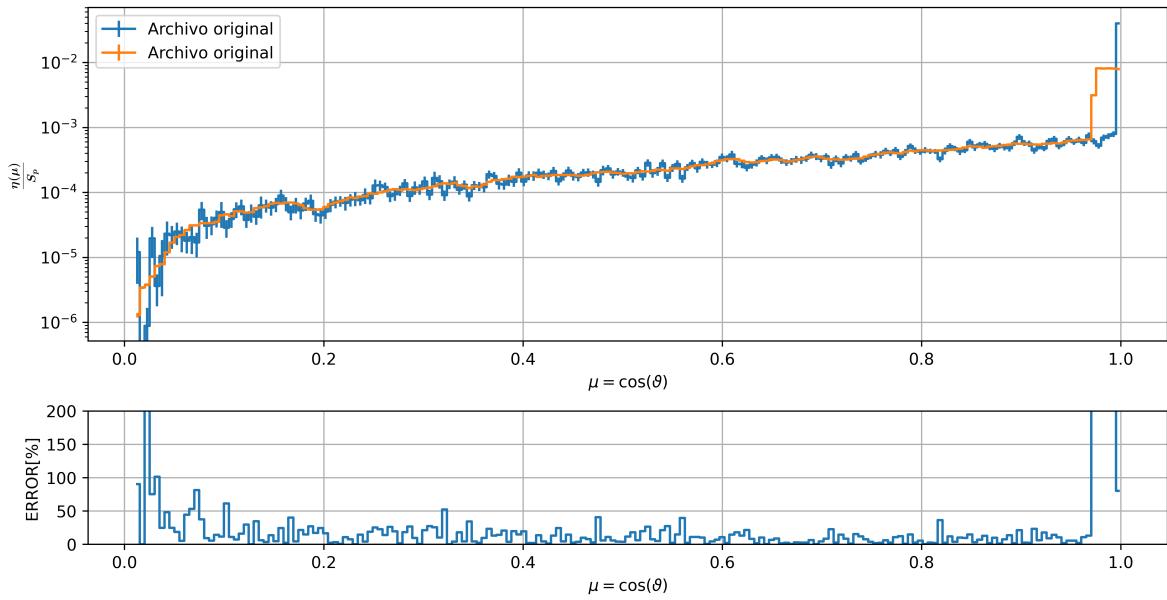


Figura 4.11: Distribución remuestreada de μ para el caso 2 de *bineado* uniforme con borde manual en $\mu \approx 1$.

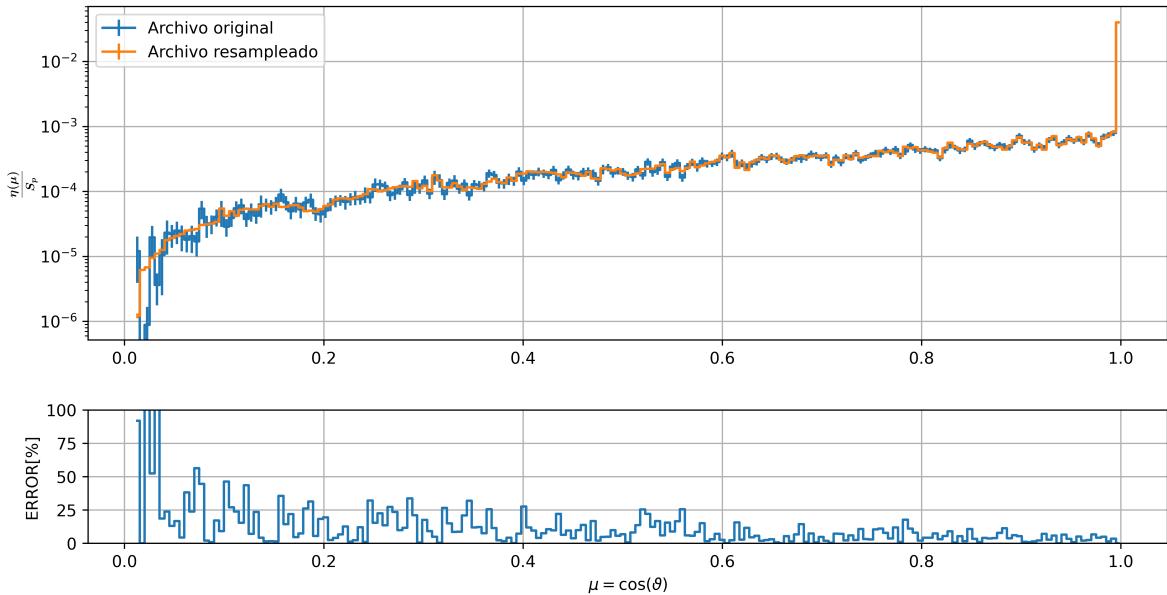


Figura 4.12: Distribución remuestreada de μ para el caso 3 de *bineado* adaptativo sin bordes manuales.

En la Figura 4.13, se incorpora un borde manual en la zona colimada, pero no se observan cambios relevantes con respecto al caso anterior. Esto evidencia que el algoritmo adaptativo es capaz de detectar y segmentar adecuadamente esta región sin intervención adicional, siempre que la resolución no sea excesiva.

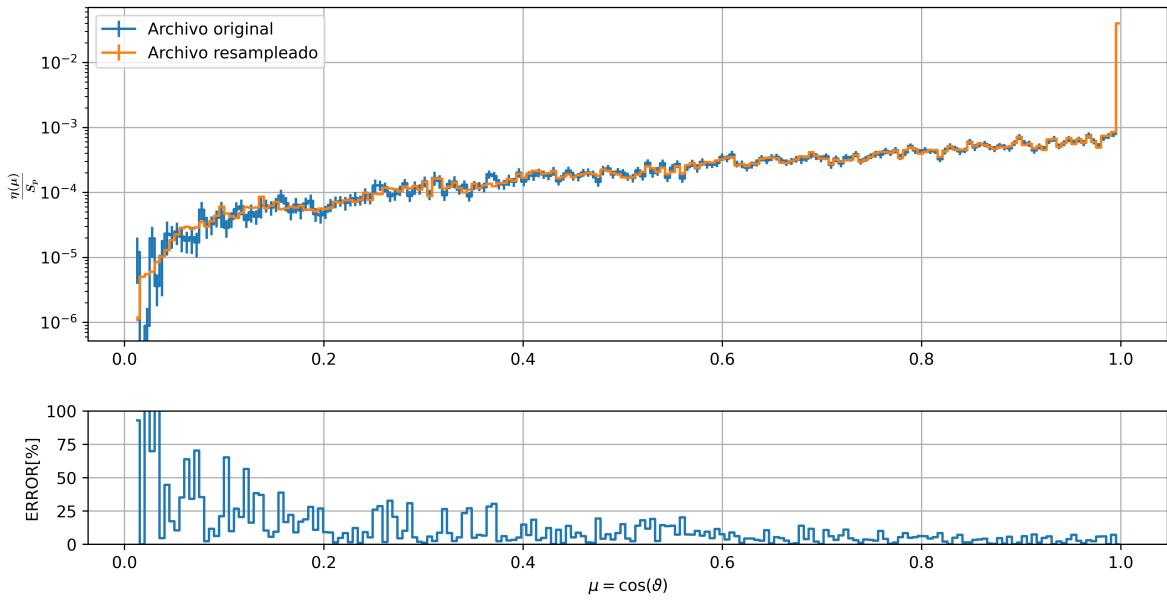


Figura 4.13: Distribución remuestreada de μ para el caso 4 de *bineado* adaptativo con borde manual en $\mu \approx 1$.

4.3.4. Distribuciones espaciales $X-Y$

En esta sección se detallan las distribuciones espaciales en el plano $X-Y$ para las cuatro configuraciones analizadas. Se evalúa particularmente la capacidad de cada método para representar correctamente la geometría del sistema, especialmente en torno a la interfaz agua–vacío definida por el tubo.

En la Figura 4.14, correspondiente al caso 1 de *bineado* uniforme sin bordes, se observa un suavizado de la estadística entre regiones físicamente distintas. En particular, aparecen partículas remuestreadas en el agua que originalmente deberían haberse generado dentro del tubo de vacío. Este efecto se debe a que los histogramas uniformes no capturan la interfaz geométrica del sistema: un mismo bin puede contener partículas tanto dentro como fuera del tubo, y al remuestrear, se reparten en ambas regiones.

En la Figura 4.15, se incorporan bordes manuales para delimitar la región correspondiente al canal de vacío. Este ajuste mejora la segmentación espacial, concentrando la estadística dentro del tubo y evitando que partículas aparezcan en el agua. Sin embargo, se observa una estructura cuadriculada en el agua, debido a la baja estadística original combinada con la discretización forzada en grupos debido al *bineado* uniforme de los histogramas macro.

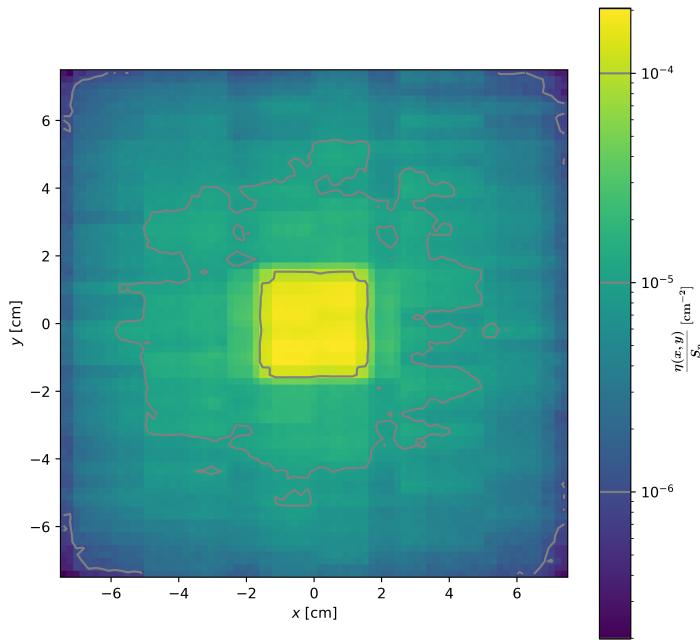


Figura 4.14: Distribución remuestreada en el plano X - Y para el caso 1 de *bineado* uniforme sin bordes manuales.

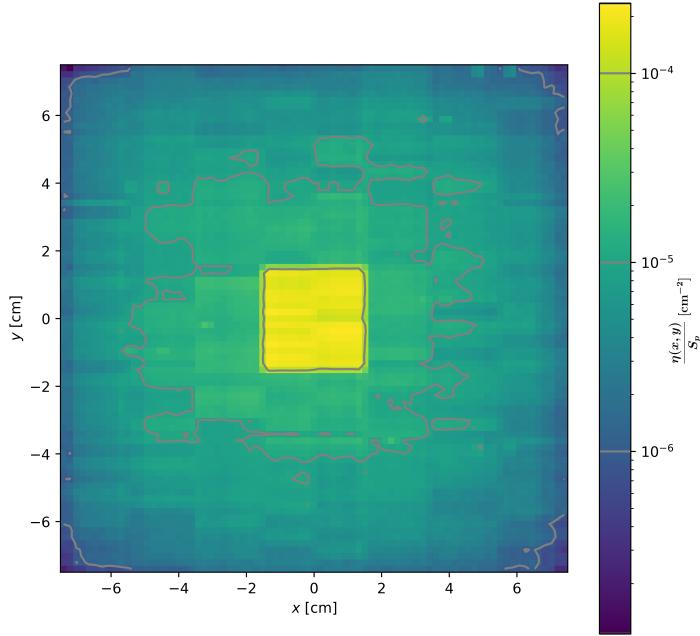


Figura 4.15: Distribución remuestreada en el plano X - Y para el caso 2 de *bineado* uniforme con bordes manuales en la interfaz agua–vacío.

La Figura 4.16 presenta el resultado del caso 3 de *bineado* adaptativo sin bordes definidos por quien utiliza el código. En este caso, se observa que el método logra capturar adecuadamente la transición entre regiones, evitando la aparición de partículas en zonas del agua donde no correspondía. La distribución dentro del agua se muestra más homogénea que en el caso anterior, debido a que el algoritmo adaptativo no fuerza

la segmentación en regiones con baja estadística.

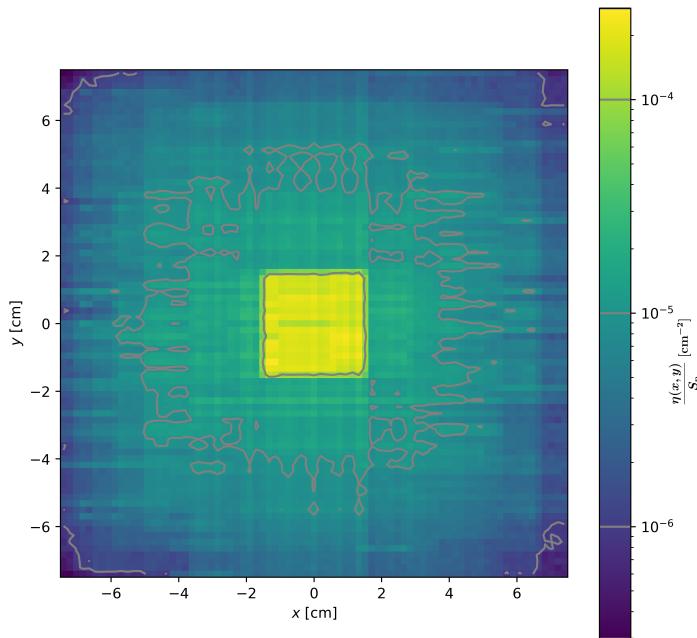


Figura 4.16: Distribución remuestreada en el plano X - Y para el caso 3 de *bineado* adaptativo sin bordes manuales.

En la Figura 4.17, se observa el caso 4 de *bineado* adaptativo con bordes definidos por quien utiliza el código en la interfaz. Si bien los resultados son visualmente similares a los del caso sin bordes, puede señalarse que aquí los límites del canal de vacío están representados de forma más precisa, dado que se han especificado explícitamente. No obstante, el método adaptativo sin bordes ya proporciona una aproximación razonable, por lo que la mejora en este caso es leve.

4.3.5. Análisis preliminar de los resultados

El análisis realizado en esta sección permitió evaluar las cuatro configuraciones de *bineado*. Se evidenció que los histogramas adaptativos ofrecen una representación más fiel del comportamiento original, incluso en ausencia de bordes manuales, en comparación con los esquemas de *bineado* uniforme.

Sin embargo, también se observó que la incorporación de bordes definidos por quien utiliza el código continúa siendo una herramienta útil cuando se dispone de conocimiento previo de la fuente. Estos bordes permiten guiar al algoritmo, segmentando zonas relevantes, lo que facilita asignar mayor resolución a otras zonas del espacio de fases.

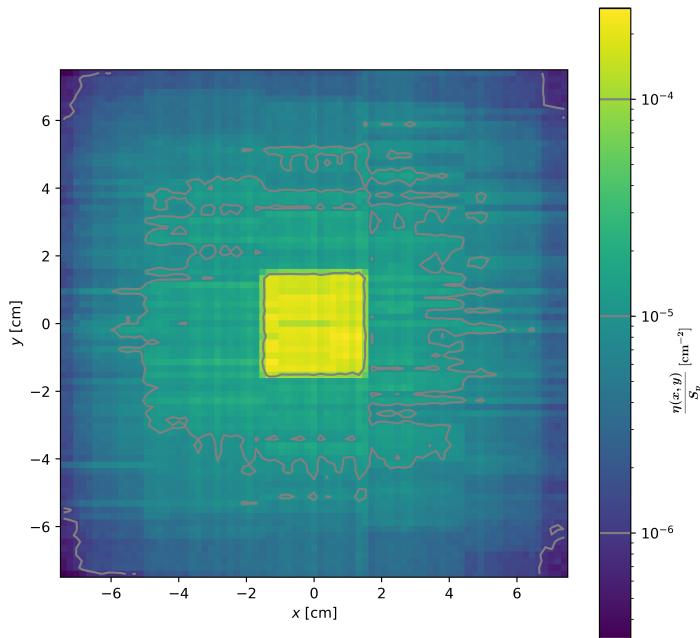


Figura 4.17: Distribución remuestreada en el plano X – Y para el caso 4 de *bineado adaptativo* con bordes definidos por quien utiliza el código.

Por otro lado, se identificaron limitaciones en el esquema adaptativo. En particular, el análisis de la variable direccional μ —cuarta en el orden de las variables— evidenció un exceso de bines debido a la baja cantidad de partículas en los subconjuntos. Esta excesiva resolución produce la reproducción del ruido estadístico en lugar de su suavizado.

Para mitigar este efecto, se propone continuar con el uso del *bineado adaptativo* con bordes manuales y reducir progresivamente la cantidad de bines de los histogramas tanto macro como micro a medida que se avanza en el orden de segmentación. Este ajuste permitiría conservar la capacidad de detección fina en las primeras variables, mientras se evita la sobresegmentación en etapas siguientes donde la estadística es más limitada.

En la siguiente sección, se aplicará esta estrategia a modo de configuración definitiva: se volverá a procesar el archivo de partículas original empleando un esquema de segmentación jerárquica con reducción progresiva de la cantidad de bines.

4.4. Procesamiento optimizado del archivo de partículas

Con base en los resultados del procesamiento preliminar, se definió una configuración optimizada para generar la fuente distribucional. El objetivo fue preservar la fidelidad de la representación estadística, evitando al mismo tiempo la sobresegmentación de los subconjuntos que se observó en configuraciones anteriores.

Para minimizar la fragmentación del conjunto de datos, se optó por utilizar la menor cantidad posible de macrogrupos, manteniendo el orden de las variables usado previamente. La configuración seleccionada fue:

$$n_{\text{macro}} = [3, 5, 3, 3]$$

Este esquema permitió trabajar con subconjuntos de mayor estadística en las etapas siguientes del procesamiento, mejorando así el procesamiento de los histogramas micro.

Asimismo, se eligió una cantidad reducida de microgrupos que garantizara una representación adecuada con un nivel de suavizado compatible con la estadística disponible. El esquema utilizado fue:

$$n_{\text{micro}} = [30, 10, 9, 16, 4]$$

Este conjunto es decreciente con el orden de segmentación, excepto en el caso de μ , donde se incrementó el número de bines para mejorar la resolución en la zona cercana a $\mu \approx 0$.

La Figura 4.18 muestra la distribución remuestreada de letargía obtenida con esta configuración. Se observa una representación precisa de la delta correspondiente a la letargía mínima, así como una buena resolución en la región de termalización. En la zona intermedia, donde la densidad estadística es menor, el suavizado logrado resulta adecuado y coherente con la tendencia de la distribución original.

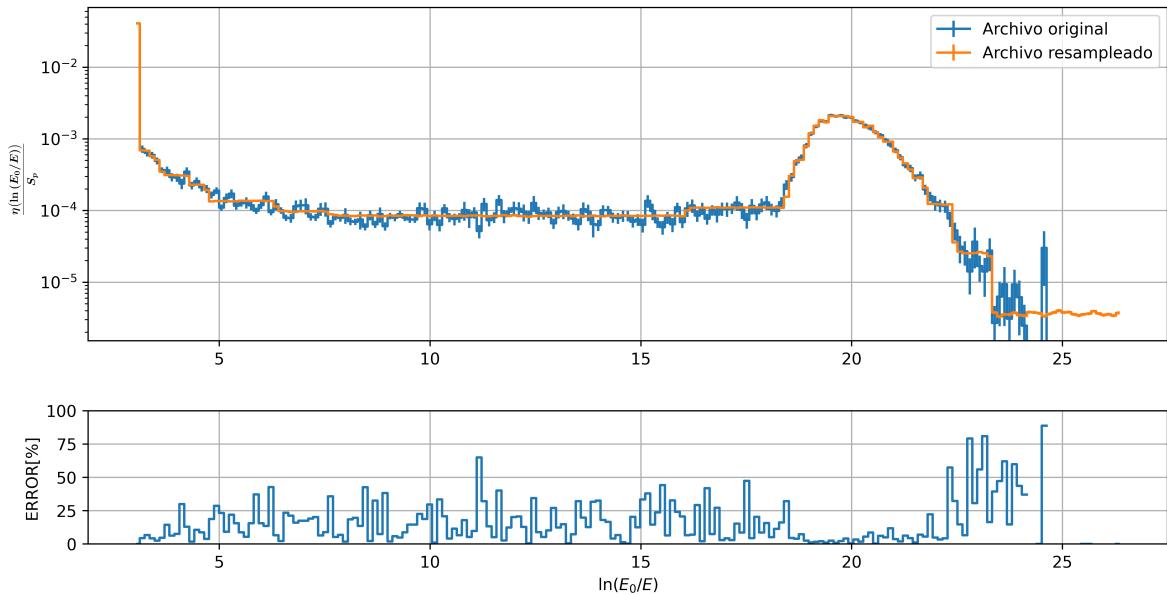


Figura 4.18: Distribución remuestreada de letargía utilizando la configuración optimizada de macro y microgrupos.

En la Figura 4.19 se muestra la distribución remuestreada de μ . Se obtiene una

correcta representación del pico en $\mu = 1$, asociado a neutrones colimados, así como un buen seguimiento en el resto del dominio, sin evidencia de ruido estadístico excesivo.

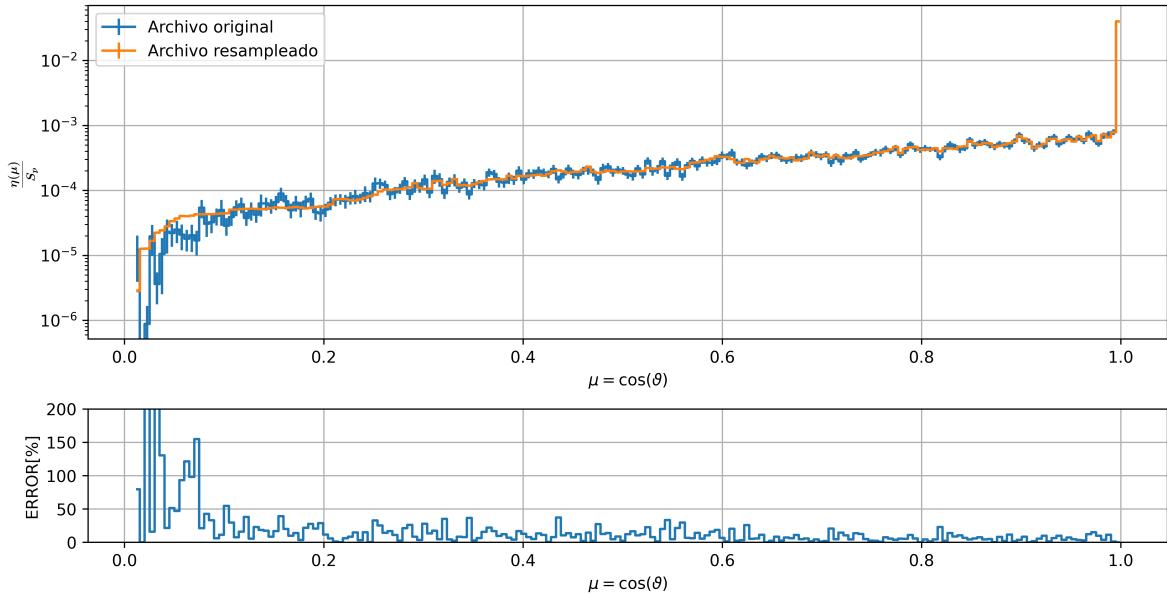


Figura 4.19: Distribución remuestreada de μ utilizando la configuración optimizada.

La Figura 4.20 presenta la distribución espacial en el plano $X-Y$. Se observa una representación precisa del canal de vacío, sin generación espuria de partículas en el agua. Sin embargo, la resolución en la región del agua es inferior a la alcanzada en configuraciones con mayor cantidad de macrogrupos. Este efecto refleja el compromiso asumido: evitar segmentar en exceso regiones con baja estadística para preservar la calidad en el procesamiento de las variables subsiguientes.

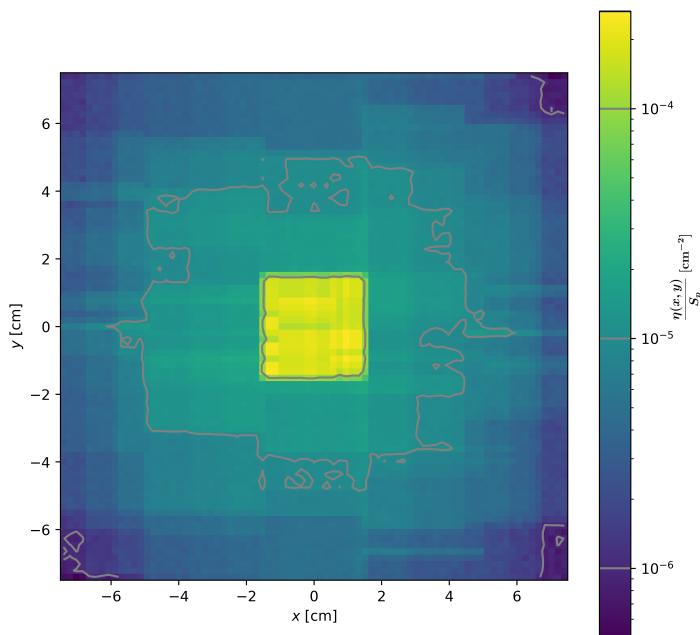


Figura 4.20: Distribución remuestreada en el plano $X-Y$ con configuración final optimizada.

La configuración descrita será utilizada como base para realizar la simulación definitiva desde la superficie de acople en adelante. En la próxima sección se presentan los resultados obtenidos y se comparan contra la simulación original de referencia, ejecutada con una mayor cantidad de partículas.

4.5. Resultados de la simulación comparativa

Para validar los resultados se evaluaron distintas magnitudes físicas a lo largo del eje del sistema:

- Flujo escalar a lo largo del tubo: en agua, y en vacío.
- Espectro energético sobre una superficie a $z = 80$ cm.
- Corriente sobre una superficie a $z = 80$ cm.

A su vez se aplicó el método de reducción de varianza de ventanas de peso implementado en OpenMC para mejorar la estadística en el agua.

4.5.1. Comparación del perfil de flujo a lo largo del canal

Con el objetivo de validar la fuente distribucional generada, se realizó una simulación de referencia desde el inicio del conducto, utilizando una cantidad elevada de partículas para asegurar una buena convergencia estadística, según se mostró previamente en la Figura 4.1. A partir de esta simulación se obtuvo el perfil de flujo original a lo largo del canal.

Posteriormente, se utilizó la fuente distribucional construida en la sección anterior y se ejecutó una nueva simulación iniciando desde la superficie ubicada a $z = 15$ cm. Esta simulación fue llevada adelante hasta alcanzar también una buena convergencia, permitiendo así una comparación directa entre ambos resultados.

La Figura 4.21 muestra los perfiles de flujo escalar normalizados por neutrón de fuente (S_p) obtenidos en ambas regiones del conducto: la región de agua moderadora (4.21a) y el canal de vacío (4.21b), junto con el error relativo porcentual.

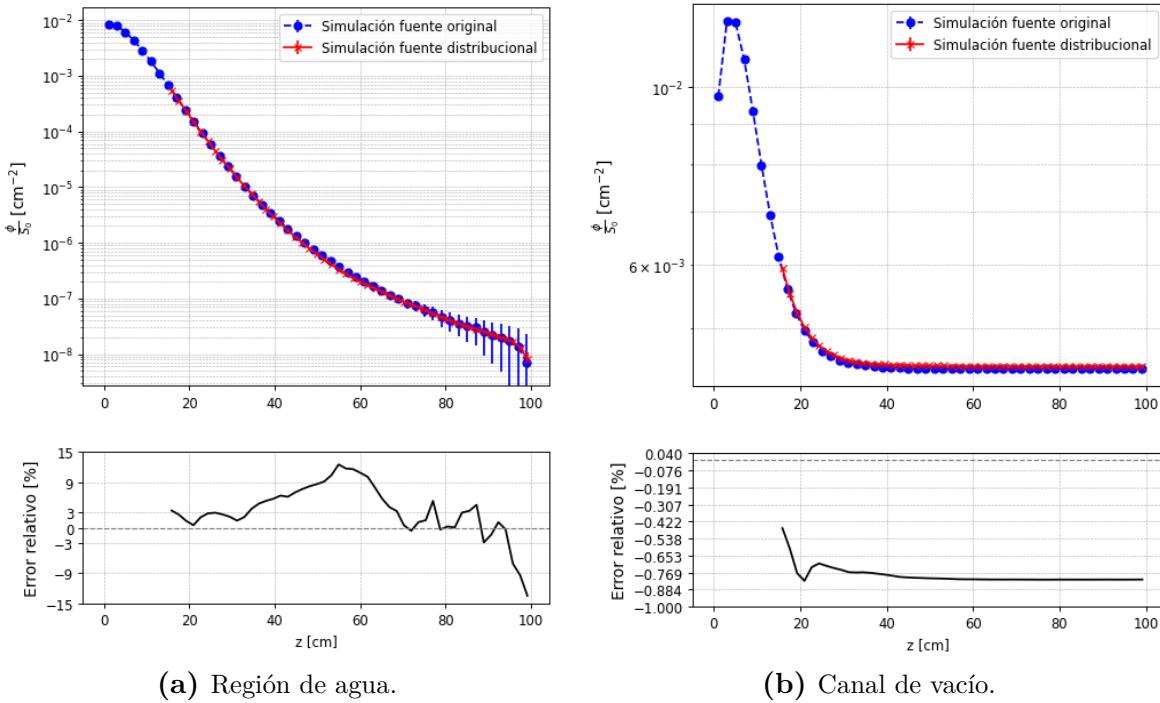


Figura 4.21: Comparación entre los perfiles de flujo escalar obtenidos mediante la fuente original (simulación larga desde el inicio del conducto) y mediante la fuente distribucional (simulación desde la superficie intermedia). En los casos donde no se aprecia las barras de error se debe a que son menores que el tamaño del símbolo. En ambas regiones se muestra también el error relativo porcentual.

En la región de agua (Figura 4.21a), se observa un error relativo que no converge con la distancia al origen de la fuente. Este comportamiento se explica por la escasa densidad de peso estadístico presente originalmente en esta región del archivo de partículas original (ver Tabla 4.1), lo cual dificultó la reconstrucción adecuada de su distribución.

En cambio, en el canal de vacío (Figura 4.21b), se logra una reconstrucción de mayor precisión del perfil original. El error relativo se mantiene por debajo del 1% en todo el tramo analizado, lo que indica una adecuada correspondencia entre la fuente generada y el perfil de referencia. Cabe destacar que parte del error observado se debe a diferencias entre las distribuciones de corriente presentes en el archivo de partículas original —limitado en estadística— y la simulación larga de referencia. No obstante, las formas del perfil en ambas simulaciones muestran concordancia, validando el enfoque implementado.

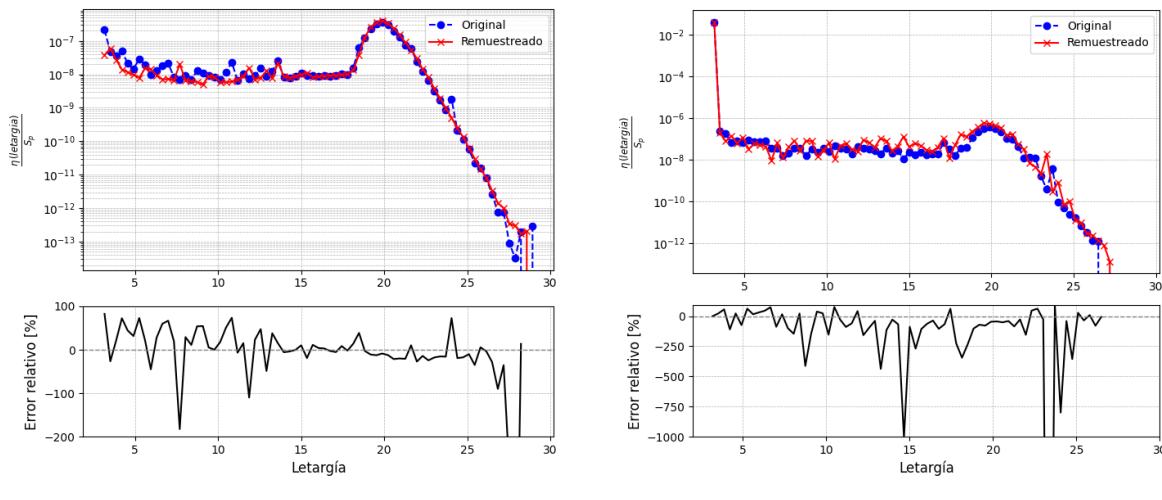
4.5.2. Comparación del espectro en la salida del canal

Además del perfil de flujo, se analizó el espectro registrado en la superficie ubicada a $z = 80$ cm. En ambas simulaciones —la original desde el inicio del conducto y la simulación utilizando la fuente distribucional— se registró el espectro de neutrones

incidentes sobre dicha superficie. La Figura 4.22 muestra los espectros normalizados por neutrón de fuente obtenidos para ambas simulaciones.

En la Figura 4.22a, correspondiente a la región de agua, se observa que la distribución en letargía generada por la fuente distribucional presenta una buena aproximación en las regiones de alta letargía, donde se encuentran los neutrones termalizados. El error relativo en esa zona es bajo, lo que indica que la fuente remuestreada logró capturar adecuadamente la estructura del espectro original.

En la Figura 4.22b se observa una correcta aproximación en la región de letargía correspondiente a $E = 1 \text{ MeV}$, donde el error relativo es muy bajo. Esto indica que el remuestreo de los neutrones colimados que se propagan por el canal de vacío se realizó correctamente. Este resultado está en concordancia con lo observado previamente en el perfil de flujo en dicha región (ver Figura 4.21b), donde el error se mantuvo por debajo del 1 %. En cambio para regiones de mayor letargía, correspondientes a neutrones que han colisionado y se propagan por la región de vacío, se observa un error relativo mayor.



(a) Región de agua.

(b) Canal de vacío.

Figura 4.22: Comparación de la distribución de letargía normalizada por neutrón de fuente (S_0) en la superficie a $z = 80 \text{ cm}$ para las simulaciones original y remuestreada. En ambos casos se incluye también el error relativo porcentual.

4.5.3. Comparación de la corriente

Se analizó la corriente total de neutrones atravesando la superficie ubicada a $z = 80 \text{ cm}$ para ambas simulaciones. En la simulación original, ejecutada desde el inicio del canal con alta estadística, se obtuvo una corriente normalizada de

$$J_{\text{original}} = (4,005 \pm 0,003) \times 10^{-2} \text{ neutrones por neutrón fuente.}$$

Por otro lado, la simulación realizada con la fuente remuestreada arrojó un valor

de

$$J_{\text{remuestreada}} = (4,037 \pm 0,008) \times 10^{-2} \text{ neutrones por neutrón fuente.}$$

Esto representa una diferencia relativa del 0.8 % del valor original, lo cual indica una aceptable conservación de la corriente total.

A su vez, en la Tabla 4.2 se presenta un análisis de la corriente de neutrones registrada en la superficie ubicada en $z = 80$ cm, subdividida en distintos subconjuntos del archivo de partículas. Para construir esta tabla, se dividió espacialmente el dominio en dos regiones: agua y vacío. Para cada una de estas regiones, se consideró tanto el conjunto total de partículas sin segmentación energética, como también los subconjuntos correspondientes a los intervalos de letargia $[0, , 4]$ y $[4, , \infty]$. A su vez, dentro de cada subconjunto energético, se realizó una división adicional en función de μ : se incluyó tanto el total (sin discriminar por dirección), como los subintervalos $\mu \in [0, , 0,5]$ y $\mu \in [0,5, , 1]$.

En la tabla se informan los valores de corriente para cada uno de estos subdominios, tanto para la simulación original como para la simulación utilizando la fuente distribucional. Se observa que las mayores diferencias relativas aparecen en aquellas regiones donde la corriente presenta mayor error estadístico, producto de la baja cantidad de partículas. En particular, esto se manifiesta en los subgrupos de baja letargia dentro de la región de agua, lo cual es esperable dado que, por su naturaleza moderadora, el agua contiene una mayor proporción de neutrones térmicos en comparación con los rápidos o epítérmicos.

Por otro lado, en la región de vacío se observa que el subdominio de letargia alta muestra una mayor diferencia relativa que el de letargia baja. Esto también es coherente, ya que en el interior del tubo de vacío predominan los neutrones provenientes directamente de la fuente, los cuales tienen baja letargia. Además, los subconjuntos correspondientes a valores bajos de μ presentan mayores errores que los restantes. Esto es esperable, dado que dichos valores corresponden a neutrones que viajan con trayectorias no paralelas al eje del tubo de vacío, y por tanto son menos numerosos. La mayoría de los neutrones en esta región se encuentran colimados, es decir, con μ cercanos a 1.

A partir de estos resultados, puede concluirse que el método desarrollado es capaz de reproducir la corriente de neutrones con una diferencia relativa menor al 5 % en aquellas regiones donde la estadística es suficiente. En cambio, en subdominios con escasa cantidad de partículas, el error estadístico asociado se incrementa notablemente, lo que se traduce en diferencias más significativas respecto a la simulación de referencia.

Rango de integración			Original J_z^+ [n/s]	Remuestreado J_z^+ [n/s]	Diferencia [%]
Región	Letargía	μ			
Agua	[0, 4]	Total	$2,74(8) \times 10^{-6}$	$2,6(2) \times 10^{-6}$	-5,1
		[0, 0,5]	$5,6(3) \times 10^{-7}$	$5,8(6) \times 10^{-7}$	3,6
		[0,5, 1]	$2,17(7) \times 10^{-6}$	$2,0(2) \times 10^{-6}$	-7,8
	[4, ∞]	Total	$2,9(5) \times 10^{-7}$	$1,2(8) \times 10^{-7}$	-59
		[0, 0,5]	$6,2(5) \times 10^{-9}$	$3(3) \times 10^{-9}$	-52
		[0,5, 1]	$2,9(5) \times 10^{-7}$	$1,2(8) \times 10^{-7}$	-59
	[4, ∞]	Total	$2,44(5) \times 10^{-6}$	$2,5(2) \times 10^{-6}$	2,5
		[0, 0,5]	$5,628(3) \times 10^{-7}$	$5,8(6) \times 10^{-7}$	3,1
		[0,5, 1]	$1,88(6) \times 10^{-6}$	$1,9(2) \times 10^{-6}$	1,1
Vacío	[0, 4]	Total	$4,005(3) \times 10^{-2}$	$4,037(8) \times 10^{-2}$	0,80
		[0, 0,5]	$1,09(7) \times 10^{-7}$	$1,2(3) \times 10^{-7}$	10
		[0,5, 1]	$4,005(3) \times 10^{-2}$	$4,037(8) \times 10^{-2}$	0,80
	[4, ∞]	Total	$4,004(3) \times 10^{-2}$	$4,037(8) \times 10^{-2}$	0,82
		[0, 0,5]	$1,0(2) \times 10^{-9}$	$2(2) \times 10^{-9}$	100
		[0,5, 1]	$4,004(3) \times 10^{-2}$	$4,037(8) \times 10^{-2}$	0,82
	[4, ∞]	Total	$3,4(2) \times 10^{-6}$	$5,5(6) \times 10^{-6}$	62
		[0, 0,5]	$1,08(7) \times 10^{-7}$	$1,2(3) \times 10^{-7}$	11
		[0,5, 1]	$3,3(2) \times 10^{-6}$	$5,4(6) \times 10^{-6}$	64

Tabla 4.2: Comparación de corrientes para diferentes rangos de energía y ángulo para las regiones de agua y vacío.

4.6. Conclusiones preliminares

El estudio aplicado al caso de un canal de vacío rodeado por agua permitió evaluar el desempeño del método de generación de fuentes distribucionales basado en histogramas multidimensionales. Este caso representa una situación física relevante y sensible, ya que involucra la coexistencia de poblaciones de neutrones con comportamientos distintos: una colimada y monoenergética en el vacío, y otra dispersa y moderada en el agua.

Los principales resultados obtenidos son:

- Se logró reconstruir el flujo escalar y el espectro en el canal de vacío. La región colimada de neutrones sin colisiones fue correctamente representada por el método propuesto.
- Se observó que el uso de histogramas adaptativos permite una representación más equilibrada y autónoma del espacio de fases, sin necesidad de intervención manual. No obstante, la incorporación de bordes definidos por quien utiliza el código sigue siendo una herramienta útil cuando se dispone de información a priori sobre el sistema, especialmente para evitar mezclas artificiales entre poblaciones diferenciadas.
- La implementación de una reducción progresiva en la resolución de los histogramas macro y micro resultó eficaz para evitar el sobreajuste en conjuntos de poca estadística, evitando amplificar el ruido estadístico.

Capítulo 5

Validación experimental: Conducto N°5 del reactor RA-6

En el presente capítulo se validará el método de remuestreo de partículas mediante histogramas multidimensionales desarrollado en este trabajo. Para ello, se aplicará dicho método al conducto N°5 del reactor RA-6, el reactor nuclear de investigación tipo piletta ubicado en el Centro Atómico Bariloche (CAB), Argentina.

El RA-6 cuenta con cinco conductos de extracción de neutrones diseñados para transportar haces desde el núcleo hacia diferentes instalaciones experimentales. En particular, el Departamento de Neutrones del CAB ha desarrollado espectrometría basada en la técnica de tiempo de vuelo (TdV) [19] sobre el conducto N°5. Esto se realiza utilizando un componente denominado *chopper*, cuya función es pulsar el haz de neutrones. Midiendo el tiempo que tardan los neutrones en viajar desde el *chopper* hasta un banco de detectores de ^3He , se obtiene el espectro de energía del haz mediante TdV.

A su vez, el Departamento de Neutrones ha realizado simulaciones Monte Carlo del RA-6 utilizando el código `OpenMC` con el objetivo de estimar la distribución espectral en el banco de detectores. Sin embargo, debido a la baja probabilidad de que un neutrón simulado desde el núcleo alcance dicha región, una simulación directa resulta computacionalmente inviable con los recursos disponibles.

Para superar esta limitación, se empleó una segmentación geométrica del problema. En una primera simulación desde núcleo, el Departamento de Neutrones registró un archivo de partículas en la entrada del conducto N°5. Con este archivo de partículas se generó una fuente distribucional con el método de histogramas multidimensionales. Esta fuente se utilizó como entrada en una segunda simulación de `OpenMC`, que modela exclusivamente el conducto N°5 hasta el banco de detectores.

Este enfoque permite generar una mayor cantidad de partículas en la entrada del conducto N°5 de las que se registró originalmente en el archivo de partículas. Esto

permite incrementar la estadística en los detectores sin necesidad de simular desde el núcleo.

De este modo, se posibilita una comparación entre los resultados obtenidos aplicando el método de remuestreo utilizando histogramas multidimensionales y las mediciones experimentales realizadas por el Departamento de Neutrones, permitiendo validar el método implementado.

En las secciones siguientes se describen la geometría del reactor y del conducto N°5, y los resultados obtenidos de flujo neutrónico y espectro de energía.

5.1. Descripción del reactor RA-6 y instalación experimental del conducto N°5

El núcleo del reactor RA-6 está compuesto por veinte elementos combustibles del tipo placa de aluminio *meat* de siliciuro de uranio enriquecido al 19,70 %. Este conjunto, cuya altura activa alcanza los 61,9 cm, se encuentra sumergido en una pileta de agua liviana que actúa simultáneamente como moderador, refrigerante, reflector axial y blindaje biológico. Lateralmente, el reflector está constituido por bloques de grafito. El núcleo opera a una potencia de 1 MW térmico y está alojado en una pileta cilíndrica de 2,4 m de diámetro, rodeada a su vez por un blindaje biológico de hormigón pesado con forma octogonal. En la Figura 5.1 se presenta un esquema representativo del núcleo y sus alrededores.

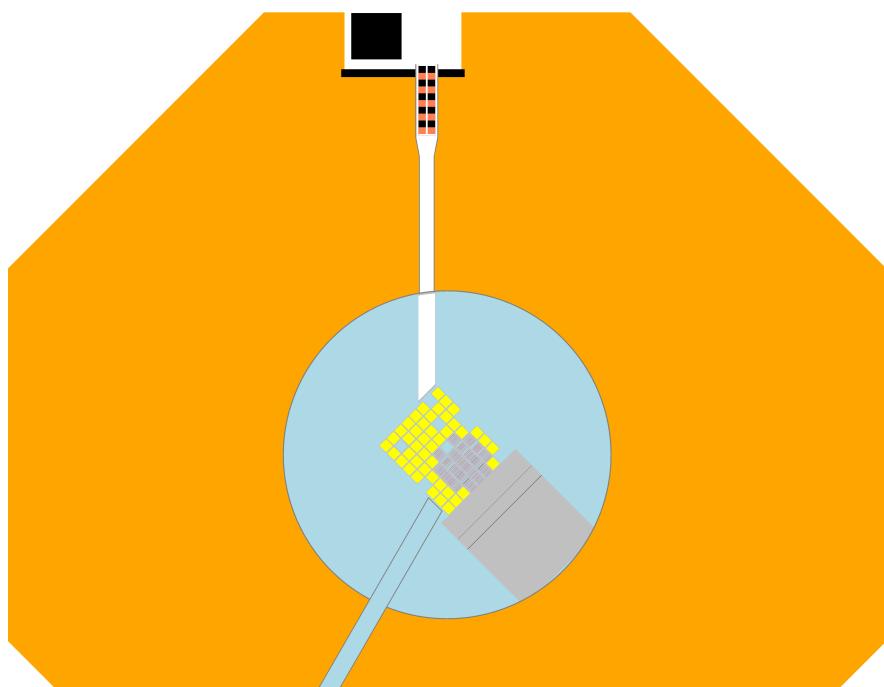


Figura 5.1: Esquema representativo del núcleo del RA-6. En el mismo se observa la disposición de los elementos combustibles, el conducto N°1 (inferior) y el conducto N°5 (superior) [19].

El reactor cuenta con cinco conductos de extracción de neutrones destinados a experimentos neutrónicos. En la Figura 5.1 se destacan dos de ellos: el conducto N°1 y el conducto N°5. En particular, el conducto N°5 se orienta hacia una zona del núcleo en la que se encuentran ubicadas un bloque de grafito y un bloque de agua. Dado que no apunta directamente hacia los elementos combustibles, el espectro resultante en este canal está compuesto mayoritariamente por neutrones térmicos.

El conducto N°5 consiste en un cilindro de acero de 5 cm de radio a la entrada que aumenta a 7,5 cm de radio en el tramo final. Este atraviesa la pileta del reactor y el blindaje biológico. En su interior, en la región de sección de 7,5 cm de radio, se encuentra un colimador constituido por secciones alternadas de plomo y parafina borada. Además, las dos primeras secciones del colimador contienen un filtro de bismuto, utilizado para atenuar la componente de radiación γ del haz. Esta atenuación resulta fundamental en experimentos, ya que permite reducir el fondo de radiación γ que puede saturar los detectores utilizados. La Figura 5.2 muestra un esquema detallado del conducto N°5 y de su sistema de colimación, como así también el banco de detectores utilizados en la medición experimental.

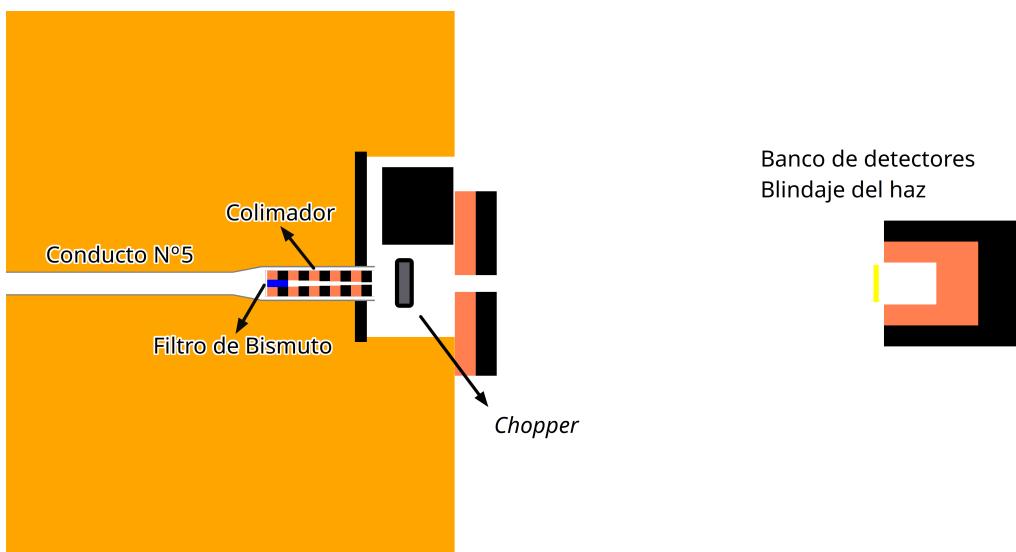


Figura 5.2: Esquema representativo del conducto N°5 del RA-6 y de la instalación experimental asociada al *chopper* [20].

Este conducto está asociado a un componente experimental denominado *chopper*, diseñado para la generación de haces pulsados de neutrones. El *chopper* consiste en un disco rotatorio de material absorbente con una ranura, el cual se posiciona a la salida del conducto. Al girar a velocidad constante, la ranura permite periódicamente el paso de neutrones cuando se encuentra alineada con el eje del haz. De esta manera, se obtienen pulsos neutrónicos que pueden ser utilizados para realizar espectrometría mediante la técnica de tiempo de vuelo (TdV). La Figura 5.3 ilustra esquemáticamente el mecanismo de funcionamiento del disco *chopper*.

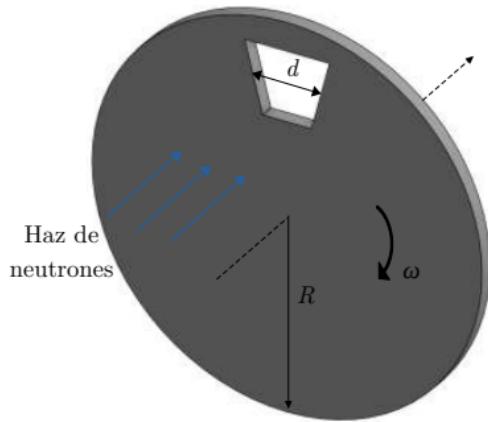


Figura 5.3: Esquema representativo del disco rotatorio del *chopper* empleado en el conducto N°5. La ranura permite el paso periódico de neutrones, generando un haz pulsado [19].

Para la implementación experimental de esta técnica, se dispone un banco de cinco detectores de ${}^3\text{He}$ a una distancia conocida de la salida del conducto. Estos detectores registran el tiempo de llegada de cada neutrón, a partir del cual se calcula su energía cinética mediante la expresión:

$$E = \frac{1}{2}m_n v^2, \quad (5.1)$$

donde E es la energía del neutrón, m_n su masa, y v la velocidad obtenida como el cociente entre la distancia conocida y el tiempo medido.

Para reconstruir correctamente el espectro neutrónico, es necesario corregir la señal detectada. En primer lugar, se debe restar un fondo constante asociado a la componente gamma y a neutrones que atraviesan el *chopper* fuera de fase. Adicionalmente, se aplican correcciones por el tiempo muerto del sistema de adquisición de datos y por la eficiencia de detección de los detectores de ${}^3\text{He}$.

Finalmente, detrás del banco de detectores se instala un blindaje adicional que permite blindar el haz y proteger las áreas experimentales circundantes.

5.2. Procesamiento del archivo de partículas

El Departamento de Neutrones del Centro Atómico Bariloche proporcionó un archivo de partículas que contiene información únicamente sobre los neutrones que ingresan al conducto N°5, filtrando los fotones. Este archivo fue generado a partir de una simulación del núcleo completo del reactor RA-6 utilizando el código *OpenMC*. La simulación consistió en un total de 10^{10} neutrones, de los cuales 41245 fueron registrados en la entrada del conducto N°5.

Cabe destacar que esta simulación fue realizada con un esquema de reducción de

varianza, lo que implica que muchas partículas del archivo presentan un peso estadístico menor a la unidad. La suma total de los pesos estadísticos de las partículas registradas fue 22182.

En la Figura 5.4 se muestra la distribución espacial de las partículas registradas en el plano XY . Puede observarse que las posiciones conforman un círculo de radio 5 cm, correspondiente a la sección transversal del conducto N°5. La distribución es aproximadamente uniforme dentro del círculo.

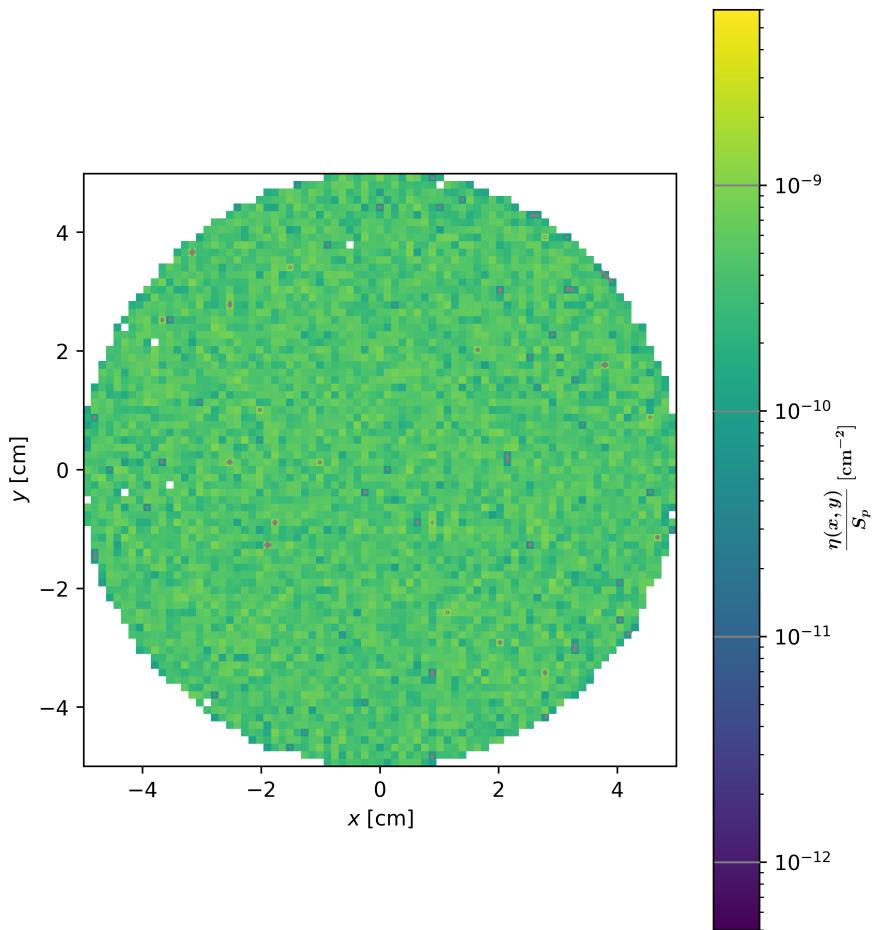


Figura 5.4: Distribución espacial de las partículas del archivo original en el plano XY .

La Figura 5.5 presenta la distribución de letargía de las partículas registradas. Se observan dos regiones de acumulación: una alrededor de $\ln(E_0/E) \approx 2$, correspondiente a energías típicas de fisión, y otra de mayor intensidad en la región térmica, centrada en $\ln(E_0/E) \approx 19$, donde $E_0 = 20$ MeV.

A partir de este archivo de partículas, se generó una fuente distribucional mediante el método de histogramas multidimensionales desarrollado en este trabajo. La configuración adoptada para la construcción de la fuente distribucional fue la siguiente:

- **Orden de procesamiento:** [letargía, X, Y, μ , ϕ]
- **Método de bineado micro y macro:** bineado adaptativo.

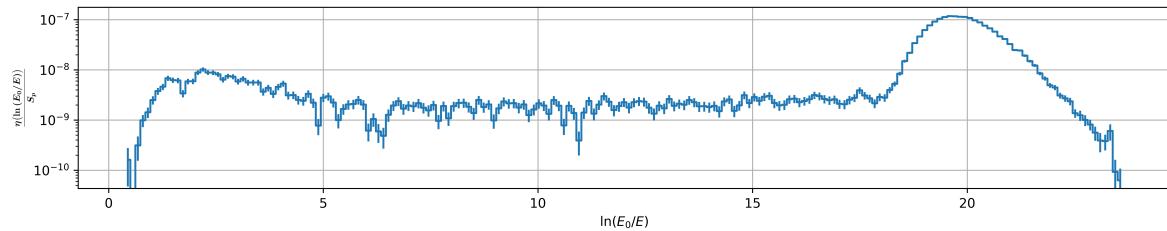


Figura 5.5: Distribución de letargía de las partículas del archivo original.

- Número de histogramas macro: [3, 7, 4, 4]
- Número de histogramas micro: [75, 18, 18, 15, 10]

Se asignó una mayor resolución a la variable letargía, dado que constituye el objetivo principal de análisis en este capítulo, centrado en la comparación espectral mediante la técnica de tiempo de vuelo. Además, se destinó una mayor cantidad de divisiones macro a la coordenada X para mejorar la representación del contorno circular del conducto. Se verificó que al reducir la resolución en dicha coordenada, aparecían patrones discretos artificiales en el plano XY , producto de una segmentación insuficiente del dominio espacial.

La Figura 5.6 muestra la distribución en el plano XY de las partículas generadas por la fuente distribucional. Puede observarse que la forma circular de radio 5 cm se reproduce adecuadamente, aunque la discretización asociada a los histogramas macro introduce estructuras visibles en los bordes.

En la Figura 5.7 se presenta la comparación entre la distribución de letargía del archivo original y la correspondiente a la fuente generada. La mayor resolución elegida para el eje de letargía permite representar en detalle la zona térmica, a costa de reproducir parte del ruido estadístico presente en la región epitérmica.

5.3. Resultados de simulación en OpenMC y comparación experimental

A partir de la fuente distribucional generada mediante histogramas multidimensionales, se llevó a cabo una segunda simulación del conducto N°5, esta vez utilizando dicha fuente como entrada en el código OpenMC. La simulación constó de un total de 5×10^9 partículas. Con el objetivo de aumentar la probabilidad de detección y mejorar la eficiencia estadística, se implementó un esquema de reducción de varianza de absorción implícita.

Durante la simulación se registró el flujo neutrónico en todo el modelo, así como el espectro de neutrones detectados en el banco de detectores. El espectro simulado fue

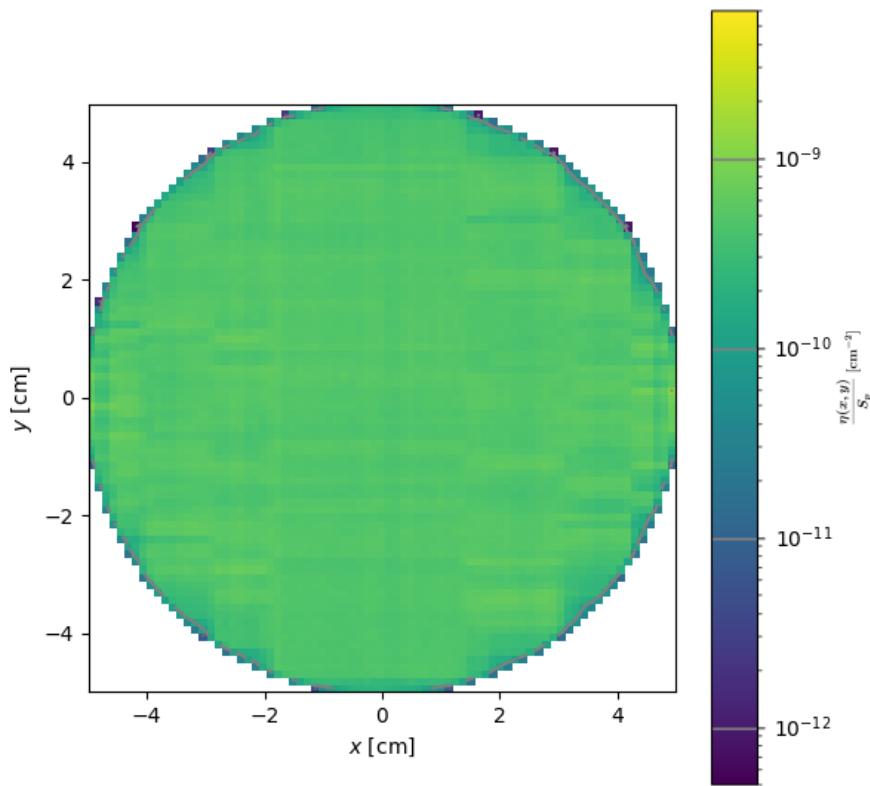


Figura 5.6: Distribución espacial de las partículas de la fuente distribucional en el plano XY .

luego comparado con los datos experimentales obtenidos mediante la técnica de tiempo de vuelo (TdV) en el RA-6 por el Departamento de Neutrones.

5.3.1. Distribución espacial del flujo

En la Figura 5.8 se presenta la distribución espacial del flujo neutrónico obtenido en el modelo del conducto N°5. Puede observarse que se logra una acumulación estadística significativa en la región del banco de detectores. Asimismo, se evidencia una fuerte atenuación del flujo a lo largo del conducto y el colimador. Esta atenuación refleja la baja probabilidad de que los neutrones atravesen el conducto y lleguen al banco de detectores.

Como se ha mostrado en trabajos relacionados sobre el conducto N°5 del RA-6 [19], simulaciones realizadas directamente desde el núcleo con cantidades de partículas del orden de 10^{10} no logran atravesar el colimador con suficiente estadística, resultando en flujos despreciables en la región de detectores. En contraste, en la Figura 5.8 puede apreciarse que, al aplicar el desacople geométrico, se ha obtenido un flujo con buena estadística en toda la zona de interés, utilizando una cantidad de partículas del mismo orden.

Este comportamiento justifica la estrategia de desacoplar geométricamente la si-

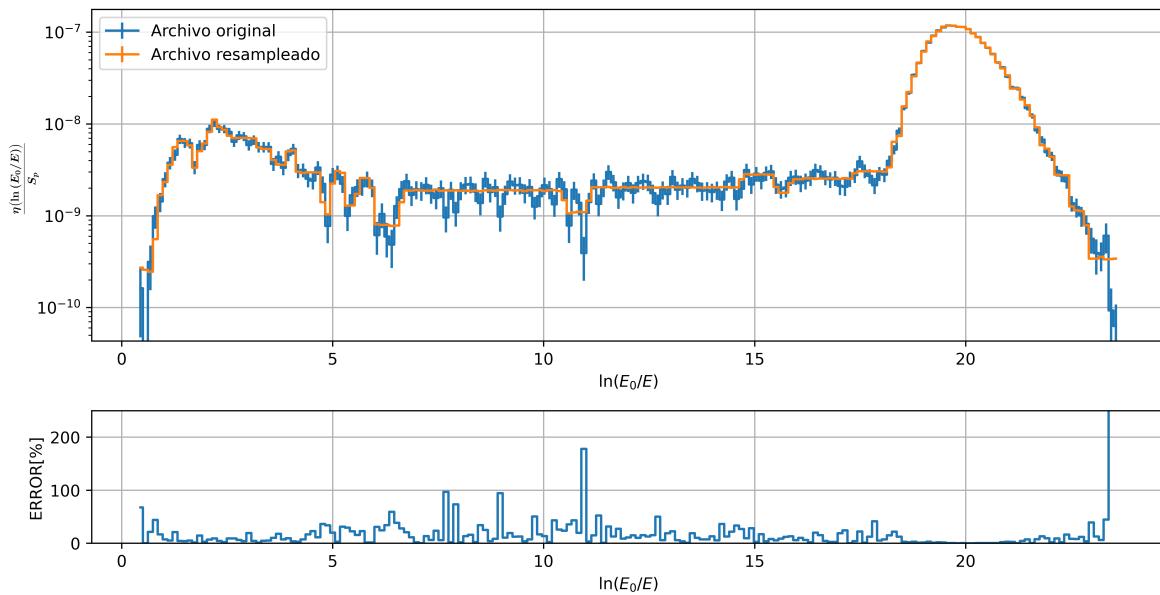


Figura 5.7: Comparación de la distribución de letargía entre el archivo original y la fuente distribucional.

mulación del conducto respecto del núcleo del reactor. Simular directamente desde el núcleo implicaría combinar la baja probabilidad de que los neutrones alcancen la entrada del conducto con la también baja probabilidad de que atravesen el colimador y lleguen al banco de detectores, resultando en una estadística reducida. En cambio, al utilizar la fuente distribucional en la entrada del conducto como fuente, se logra concentrar los recursos computacionales en la región de interés, obteniendo una estadística adecuada para el análisis espectral y mejorando los resultados.

5.3.2. Comparación con la medición experimental

La Figura 5.9 muestra el espectro neutrónico obtenido en el banco de detectores a partir de la simulación con fuente distribucional, en comparación con el espectro medido experimentalmente mediante TdV. Se observa concordancia a partir de energías del orden de 0.01 eV. No obstante, por debajo de ese umbral, el flujo simulado subestima al medido experimentalmente.

Esta discrepancia en la región de bajas energías puede atribuirse principalmente a cuatro factores:

- **Limitaciones en la representación espectral de la fuente.** Tal como se observa en la Figura 5.7, existe diferencia entre el archivo original y la fuente distribucional en la región de mayor letargía, correspondiente a las energías menores a 0.01 eV. Esta desviación se origina en la discretización inherente al método de histogramas.

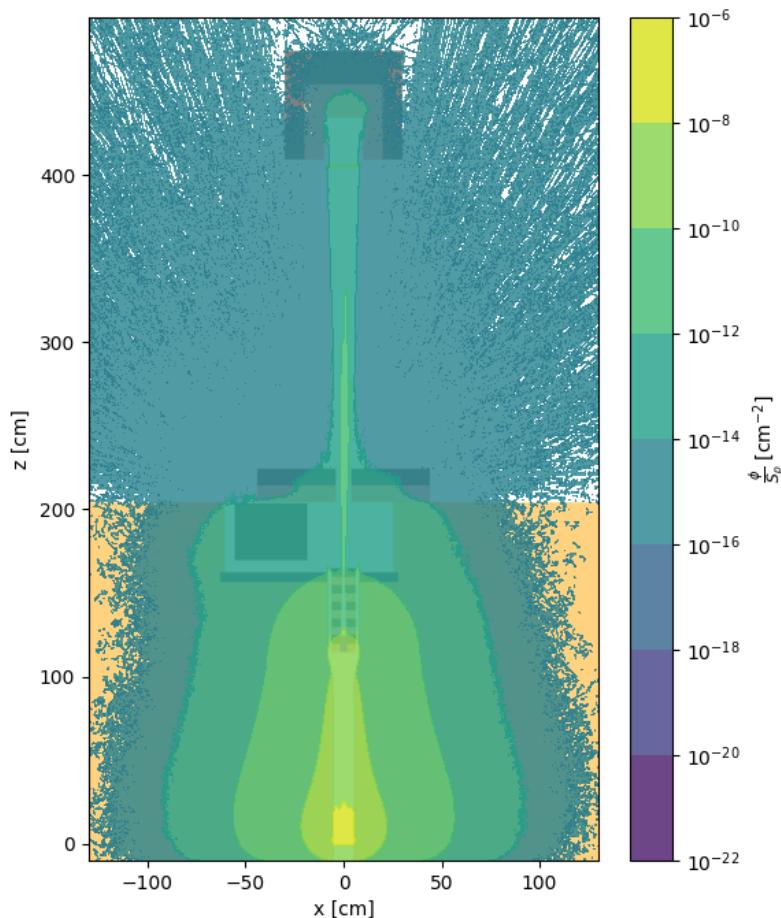


Figura 5.8: Distribución del flujo neutrónico en el modelo del conducto N°5.

- **Tratamiento del filtro de bismuto en la simulación.** En la modelización realizada con OpenMC, se emplean secciones eficaces correspondientes a átomos de bismuto como gas libre. Sin embargo, el filtro real está construido con bismuto cristalino, cuyas secciones eficaces difieren a bajas energías [21]. Mientras que para energías superiores a 0.1 eV ambas secciones eficaces son equivalentes, por debajo de ese valor la sección eficaz del bismuto cristalino es menor. La máxima discrepancia se presenta en torno a 1 meV, donde la sección eficaz del bismuto cristalino representa apenas el 10 % de la correspondiente al bismuto libre. Este efecto conduce a una atenuación del flujo térmico simulado, explicando la subestimación observada en la comparación con los datos experimentales.
- **Tratamiento experimental de los datos medidos.** La medición experimental utilizada como referencia ha sido obtenida a partir de mediciones realizadas con detectores de ${}^3\text{He}$, y requiere un proceso de postratamiento. Este proceso in-

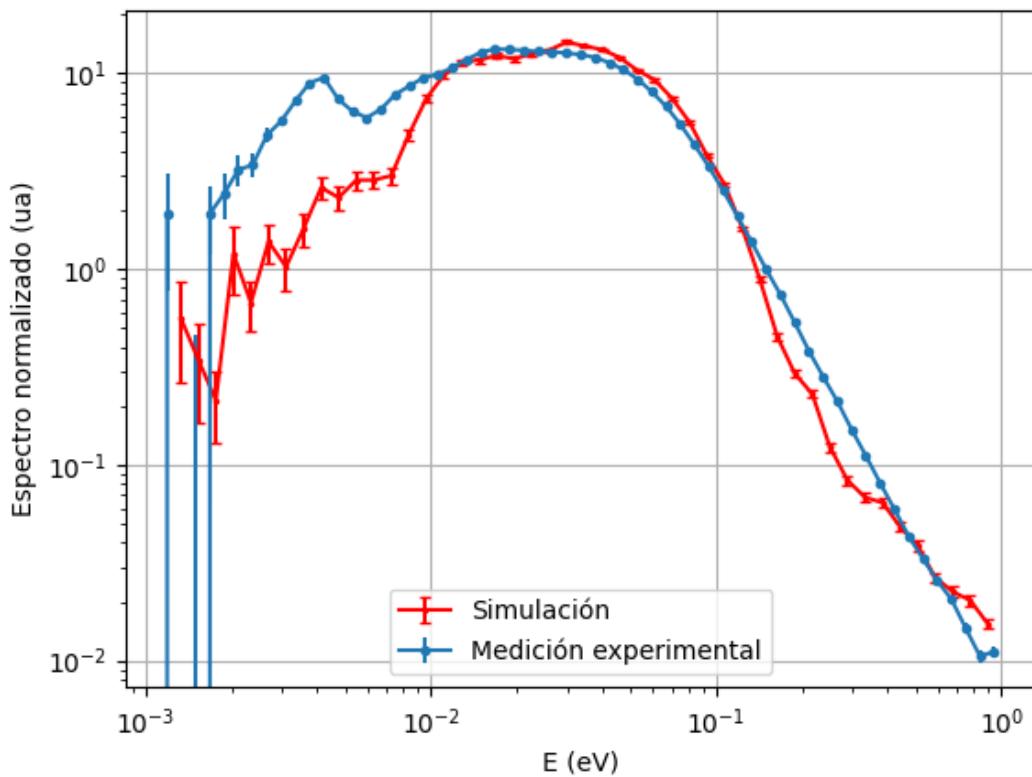


Figura 5.9: Comparación del espectro de neutrones obtenido en el banco de detectores en la simulación de `OpenMC` con la medición experimental realizada por TdV por el Departamento de Neutrones del CAB. Para ambas curvas se grafican los errores estadísticos.

cluye correcciones por tiempo muerto de los detectores, eficiencia de detección de los detectores de ${}^3\text{He}$ y por nivel de fondo. En particular, la estimación y substracción del nivel de fondo puede introducir variabilidad en la forma final del espectro, especialmente en la región correspondiente a menores energías. Como es esta la región del espectro donde se registra la mayor discrepancia respecto a la simulación, no puede descartarse que parte del desvío observado esté asociado a incertidumbres propias del procedimiento de corrección experimental aplicado.

- **Incerteza de la simulación computacional.** Toda simulación Monte Carlo está sujeta a una serie de fuentes de error que deben ser consideradas al interpretar los resultados. Entre ellas se incluyen las simplificaciones geométricas necesarias al modelar la instalación experimental, las incertidumbres en las secciones eficaces utilizadas por el código `OpenMC`, y el error estadístico asociado al número finito de partículas simuladas. Adicionalmente, el uso de un archivo de partículas finito para la construcción de la fuente distribucional introduce su propio error estadístico, que se propaga al remuestreo posterior. Finalmente, el método de generación de fuentes mediante histogramas multidimensionales introduce un

error adicional debido a la discretización del espacio de fases, que puede afectar la precisión en la reconstrucción de distribuciones continuas.

A pesar de estas limitaciones, los resultados obtenidos permiten validar el enfoque implementado, demostrando que la fuente distribucional construida es capaz de reproducir adecuadamente el comportamientopectral de los neutrones en el banco de detectores en el rango de interés térmico.

Capítulo 6

Resumen y conclusiones

En este trabajo se desarrolló e implementó un método para generar fuentes distribucionales para simulaciones Monte Carlo basado en el uso de histogramas multidimensionales. La implementación se realizó dentro del código `KDSouce`.

La metodología desarrollada fue validada mediante pruebas analíticas y contra mediciones experimentales, confirmando su capacidad para reproducir distribuciones en el espacio de fases. En particular, se mostró que la estructura de histogramas multidimensionales empleada logra preservar satisfactoriamente las correlaciones de las variables del espacio de fases de las partículas.

Adicionalmente, se implementó un esquema de *bineado* adaptativo, el cual permitió captar con precisión regiones donde las distribuciones exhiben cambios bruscos, deltas o bordes definidos, incrementando la resolución local del histograma de manera autónoma. Este esquema resultó especialmente útil en situaciones donde las distribuciones presentan estructuras complejas difíciles de caracterizar mediante discretizaciones uniformes.

Por otro lado, se realizó una implementación específica de remuestreo *on-the-fly* dentro del código fuente de `OpenMC`. Esta modificación permitió el remuestreo de partículas durante la ejecución de la simulación, sin la necesidad de generar listas intermedias de partículas. La técnica se aplicó en el caso del conducto N°5 del reactor RA-6, obteniendo resultados en concordancia con datos experimentales disponibles, validando así el método desarrollado.

Trabajo futuro

A partir de los resultados obtenidos y las capacidades mostradas por el método implementado, se identifican diversas líneas futuras de investigación que podrían extender su alcance y utilidad práctica. Una primera línea consiste en la generalización del método hacia otras geometrías más complejas y variadas, más allá de las superficies planas consideradas en este trabajo. Por ejemplo, podría emplearse una fuente distri-

buida sobre la pared cilíndrica interna de un conducto (en lugar de la superficie plana circular de entrada), de modo que se registren las partículas que atraviesan las paredes laterales y se simule únicamente su propagación hacia el exterior (útil para cálculos de blindaje o dosis).

En este mismo sentido, podría explorarse la implementación de un sistema de procesamiento basado en coordenadas polares, con el objetivo de mejorar la representación de superficies circulares. Si bien este enfoque permitiría una discretización más natural de geometrías cilíndricas, también introduce desafíos, como el tratamiento del borde angular en la transición de 0° a 360° . No obstante, este tipo de representación resultaría particularmente útil en conductos circulares, donde la implementación actual se basa en una malla rectangular con un mecanismo adicional que evita la generación de partículas fuera del contorno del círculo. En dichos casos, se realiza un remuestreo de la segunda coordenada espacial hasta obtener una partícula dentro del dominio circular deseado.

Otra línea posible de trabajo consiste en investigar el acoplamiento sucesivo de múltiples fuentes distribucionales. Esto implicaría realizar una simulación intermedia, generar una fuente distribucional sobre una primera superficie, y luego, a partir de esa simulación, registrar nuevamente una segunda superficie aún más cercana a la región de interés para construir una segunda fuente. Esta estrategia permitiría encadenar múltiples etapas de desacople, concentrando progresivamente la estadística en la zona de interés. Sería necesario estudiar las condiciones mínimas —como la cantidad de partículas y su distribución— que debe cumplir la primera superficie para garantizar que la segunda fuente generada mantenga una fidelidad aceptable y que el error acumulado no resulte limitante para la simulación.

Adicionalmente, sería valioso incorporar técnicas que permitan un refinamiento local dirigido por quien utiliza el programa. En particular, podría implementarse la capacidad de especificar manualmente un mayor número de bines en rangos específicos para ciertas variables, mejorando selectivamente la resolución de la distribución sin afectar la estructura general de macrogrupos utilizada. Esta capacidad permitiría a los usuarios optimizar el detalle local en la generación de fuentes, adaptándose de manera flexible a distintos escenarios y necesidades de simulación.

Finalmente, dado que el método desarrollado ha sido implementado únicamente para neutrones, sería relevante extender su aplicación a otras partículas, como fotones, ampliando así sus posibles aplicaciones en simulaciones Monte Carlo.

Apéndice A

Ejemplificación del *bineado adaptativo*

En este apéndice se presenta una serie de ejemplos ilustrativos que permiten visualizar el funcionamiento del algoritmo de *bineado* adaptativo descrito en el Capítulo 3. La variable seleccionada para la demostración es la letargía $\ln(E_0/E)$ (donde $E_0 = 20 \text{ MeV}$) de un archivo de partículas, que se presenta en la Figura A.1. La distribución de letargía en este caso exhibe tres regiones distintivas:

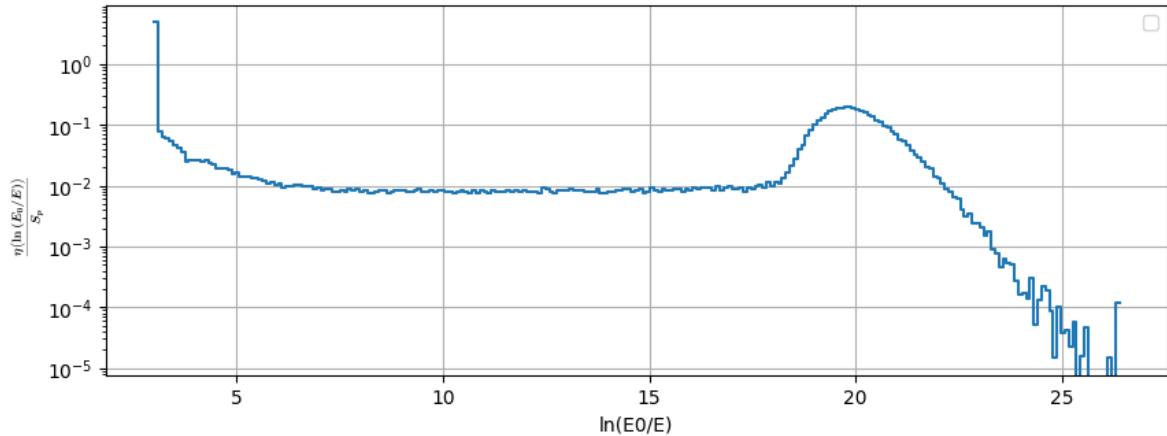


Figura A.1: Distribución de letargía utilizada en la ejemplificación.

- Una delta de Dirac en 1 MeV asociada a una fuente monoenergética.
- Una región aproximadamente constante en la región epitérmica.
- Una distribución Maxwelliana correspondiente a la moderación de los neutrones.

Se parte de la distribución original de referencia, obtenida con alta resolución producto de utilizar 50000 bines, y se aplica el algoritmo de *bineado* adaptativo siguiendo el criterio de máxima discrepancia en la función de distribución acumulada (FDC): en

cada iteración, se identifica el punto donde la diferencia absoluta entre la FDC original y la obtenida con el *bineado* actual es máxima, y se introduce allí un nuevo borde de bin. En este apéndice se utiliza 1 bin como punto de partida, y se van añadiendo bines hasta alcanzar un total de 100 bines, lo que permite observar cómo el algoritmo refina la distribución a medida que se incrementa el número de bines.

En las siguientes secciones se presentan los resultados obtenidos para distintos números de bines: 1, 2, 3, 5, 10, 25 y 100. En cada caso, se muestran:

- La distribución estimada con el *bineado* actual, comparada con la distribución de referencia.
- Las funciones de distribución acumulada (FDC) de ambas distribuciones.
- La diferencia absoluta entre ambas FDC, utilizada como criterio de refinamiento.

Caso con 1 bin

En la Figura A.2 se muestra la distribución de letargía aproximada con un único bin. Dado que el histograma se construye con un único bin, representa el valor promedio de letargía sobre todo el dominio. En la Figura A.3a se presentan las FDCs de la distribución original de referencia y la obtenida con 1 bin, mientras que en la Figura A.3b se muestra la diferencia absoluta entre ambas FDCs, que sirve como criterio para determinar la ubicación del nuevo bin en la siguiente iteración.

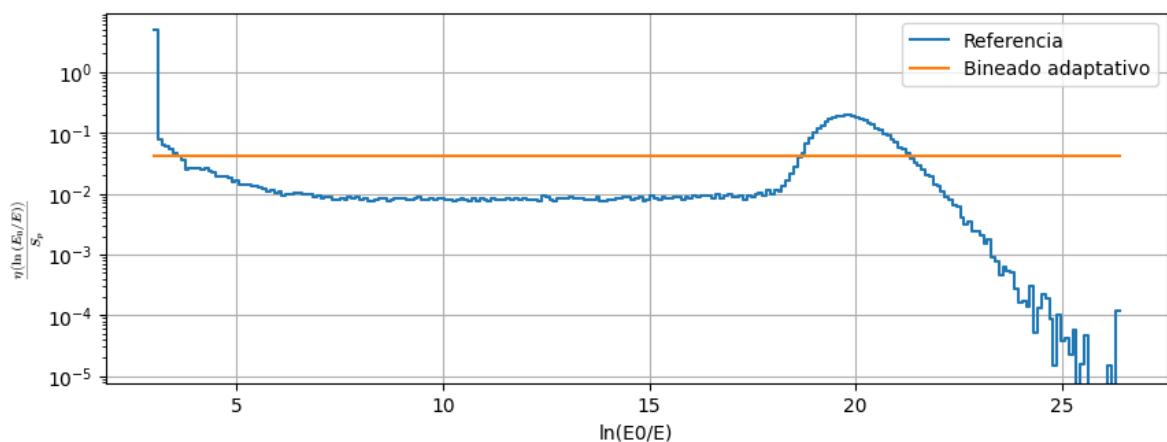
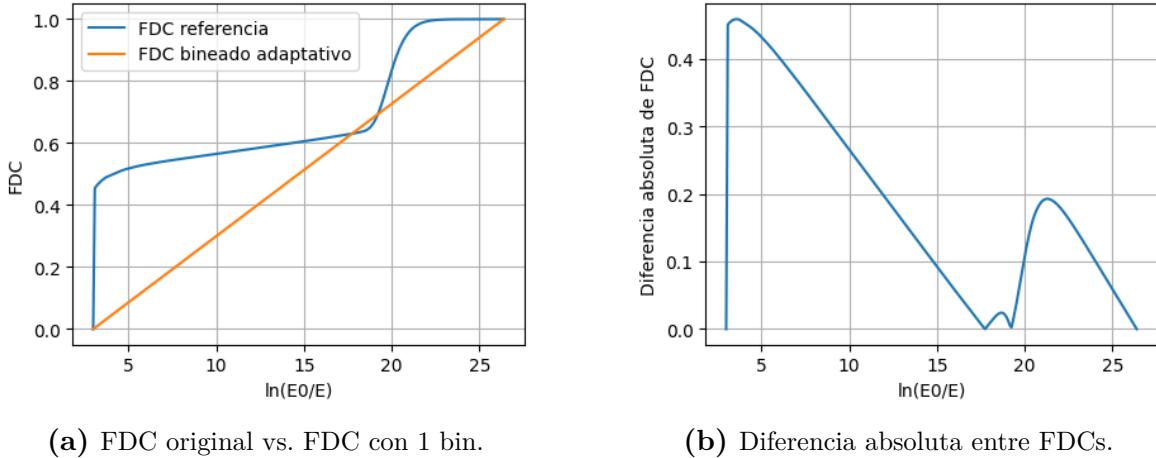


Figura A.2: Distribución de letargía aproximada con un único bin. Se observa que el histograma es constante, representando el valor promedio sobre todo el dominio.



(a) FDC original vs. FDC con 1 bin.

(b) Diferencia absoluta entre FDCs.

Figura A.3: Evaluación del criterio de refinamiento para el caso de 1 bin. Se observa que la máxima diferencia ocurre cerca de letargía ~ 3 , lo que indica que el nuevo bin debe ser ubicado en esa posición.

Caso con 2 bines

En la Figura A.4 se muestra la distribución de letargía aproximada con 2 bins. En este caso, el histograma comienza a capturar la delta en 1 MeV, pero con un bin de ancho considerable. En la Figura A.5a se presentan las FDCs de la distribución original de referencia y la obtenida con 2 bins, mientras que en la Figura A.5b se muestra la diferencia absoluta entre ambas FDCs, que sirve como criterio para determinar la ubicación del nuevo bin en la siguiente iteración. Se observa que la posición del máximo en la Figura A.3b ahora se encuentra con valor 0, producto de que se ubicó un bin en esa posición.

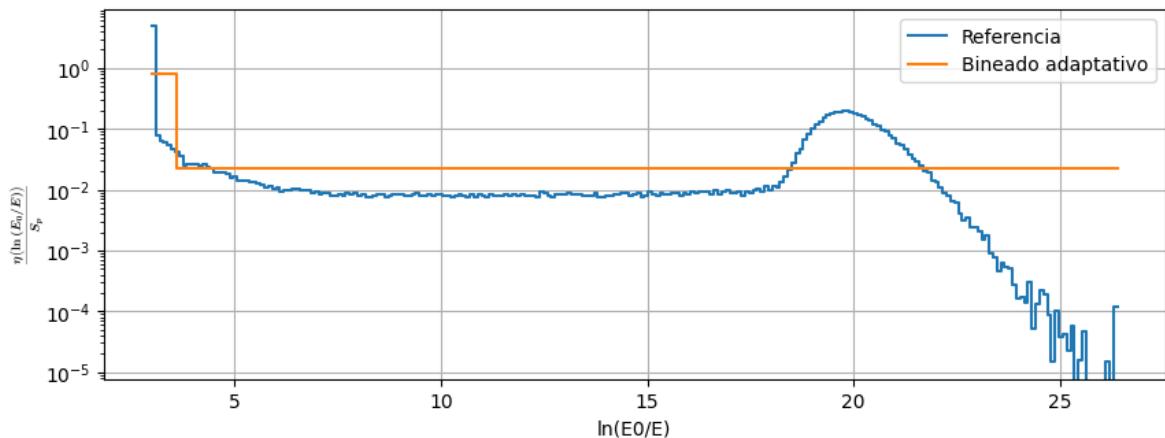
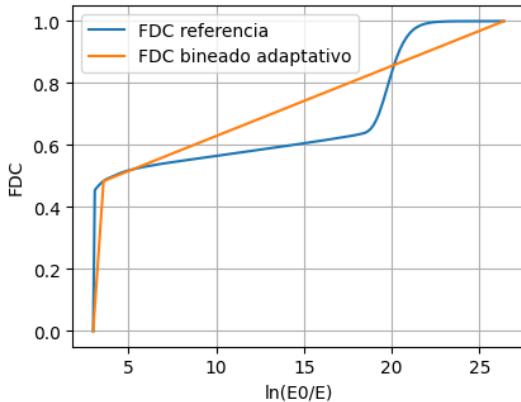
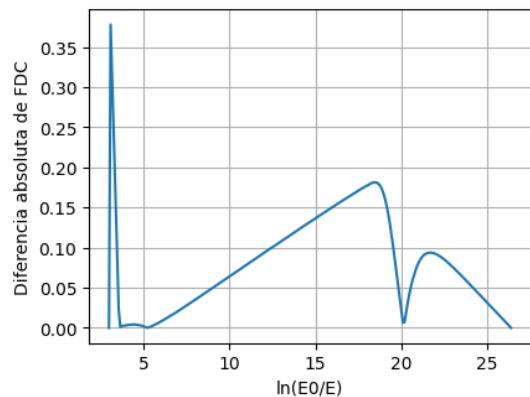


Figura A.4: Distribución de letargía aproximada con 2 bins. Se observa que el método comienza a capturar la delta en 1 MeV, pero con un bin de ancho considerable.



(a) FDC original vs. FDC con 2 bines.



(b) Diferencia absoluta entre FDCs.

Figura A.5: Evaluación del criterio de refinamiento para el caso de 2 bines. Se observa que la máxima diferencia ocurre nuevamente cerca de letargía ~ 3 , lo que indica que el nuevo bin debe ser ubicado en esa posición.

Caso con 3 bines

En la Figura A.6 se muestra la distribución de letargía aproximada con 3 bines. En este caso, el histograma logra capturar la delta en 1 MeV con mayor precisión. En la Figura A.7a se presentan las FDCs de la distribución original de referencia y la obtenida con 3 bines, mientras que en la Figura A.7b se muestra la diferencia absoluta entre ambas FDCs, que sirve como criterio para determinar la ubicación del nuevo bin en la siguiente iteración.

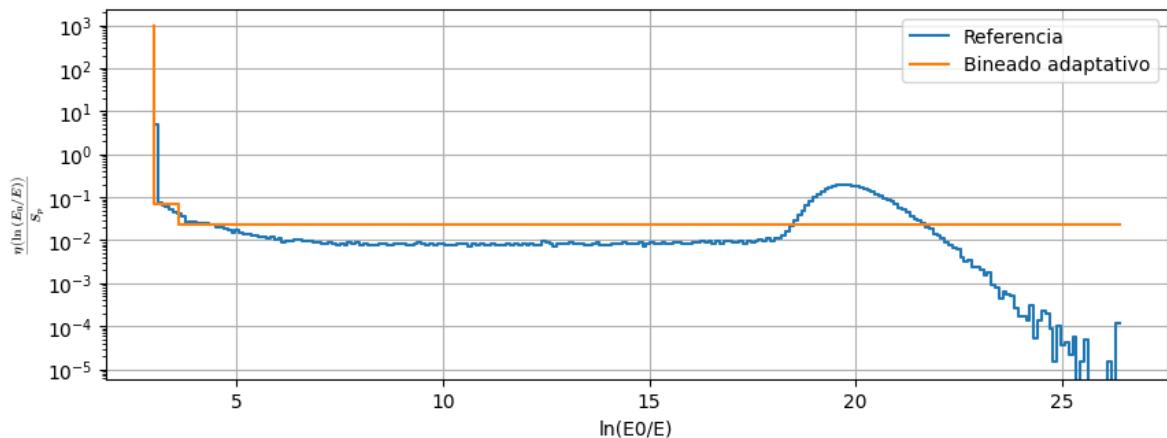
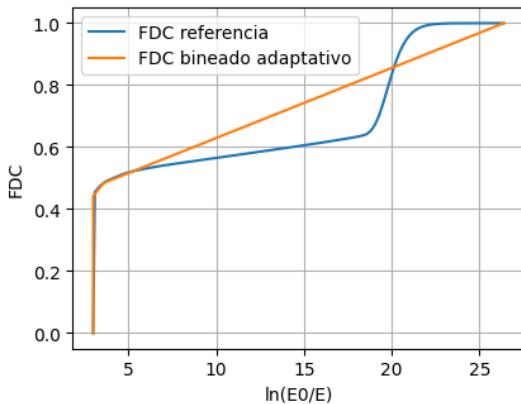
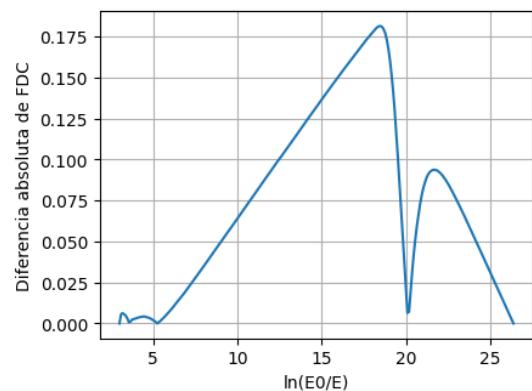


Figura A.6: Distribución de letargía aproximada con 3 bines. Se observa que el método logra capturar la delta en 1 MeV.



(a) FDC original vs. FDC con 3 bines.



(b) Diferencia absoluta entre FDCs.

Figura A.7: Evaluación del criterio de refinamiento para el caso de 3 bines. Se observa que la máxima diferencia ocurre cerca de letargía ~ 18 y letargía ~ 22 , lo que indica que los próximos dos bines deben ser ubicados en esas posiciones.

Caso con 5 bines

En la Figura A.8 se muestra la distribución de letargía aproximada con 5 bines. En este caso, el histograma comienza a capturar la distribución en la región de termalización. En la Figura A.9a se presentan las FDCs de la distribución original de referencia y la obtenida con 5 bines, mientras que en la Figura A.9b se muestra la diferencia absoluta entre ambas FDCs, que sirve como criterio para determinar la ubicación del nuevo bin en la siguiente iteración.

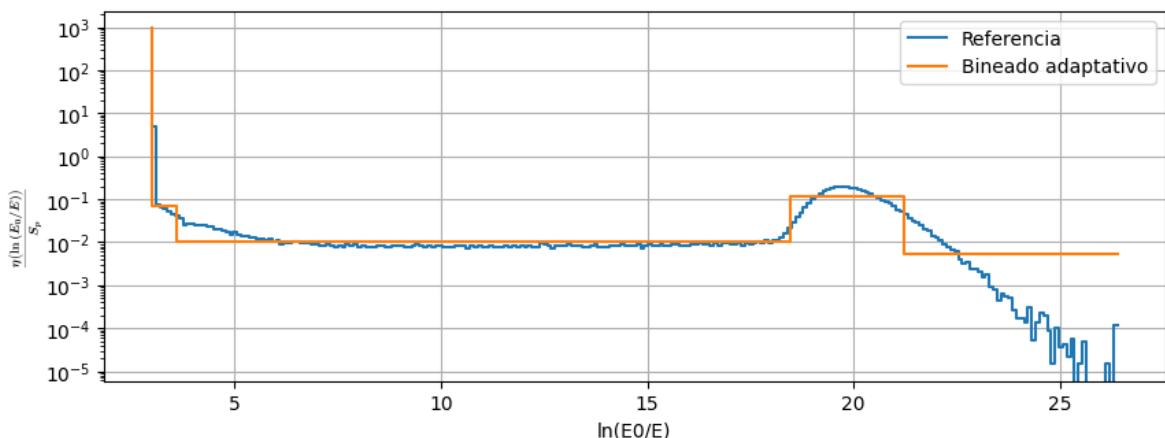
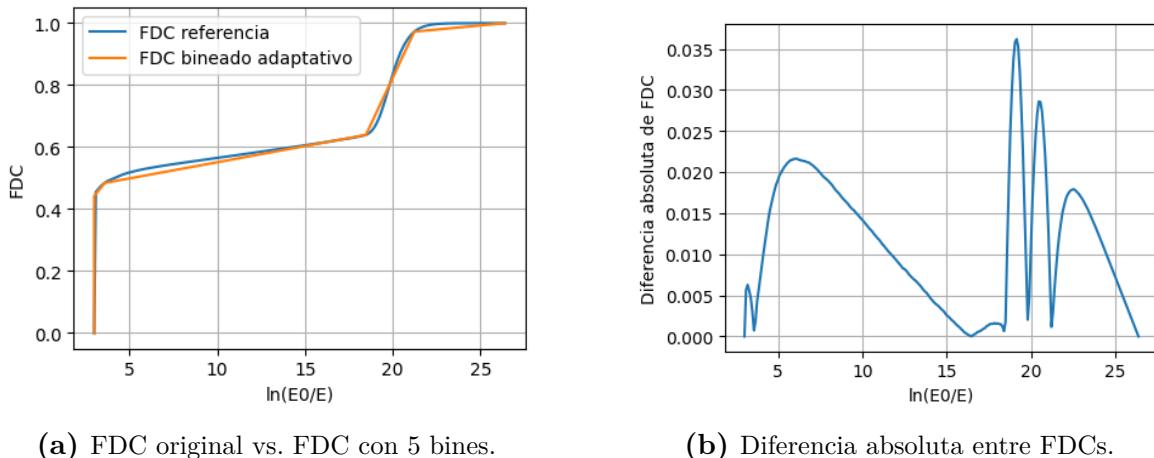


Figura A.8: Distribución de letargía aproximada con 5 bines. Se observa que el método comienza a capturar la distribución en la región de termalización.



(a) FDC original vs. FDC con 5 bines.

(b) Diferencia absoluta entre FDCs.

Figura A.9: Evaluación del criterio de refinamiento para el caso de 5 bines. Se observa en la diferencia absoluta entre FDCs los valores de mayor discrepancia, que indican que los próximos bines deben ser ubicados en esas posiciones.

Caso con 10 bines

En la Figura A.10 se muestra la distribución de letargía aproximada con 10 bines. En este caso, el histograma logra capturar la delta en 1 MeV y comienza a mejorar la aproximación de la distribución en la región de termalización y en la región epitérmica. En la Figura A.11a se presentan las FDCs de la distribución original de referencia y la obtenida con 10 bines, mientras que en la Figura A.11b se muestra la diferencia absoluta entre ambas FDCs, que sirve como criterio para determinar la ubicación del nuevo bin en la siguiente iteración.

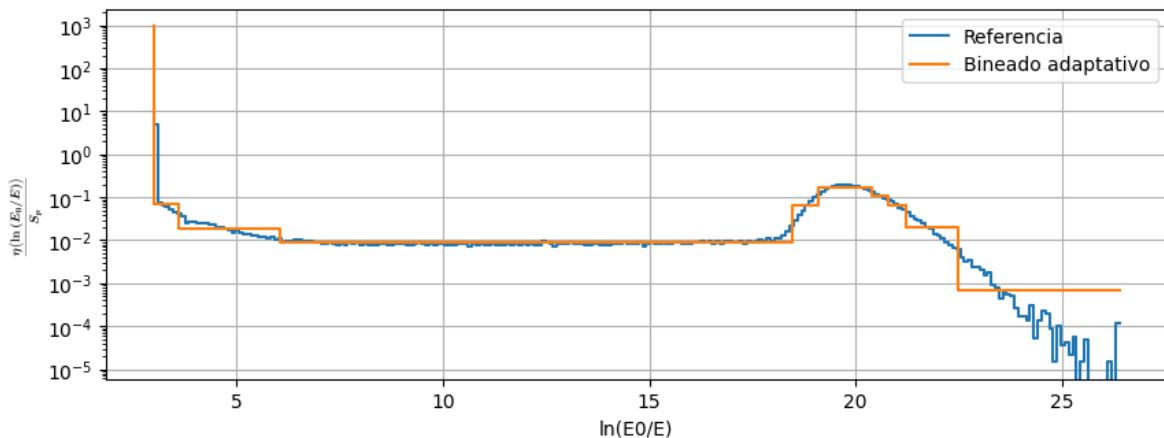
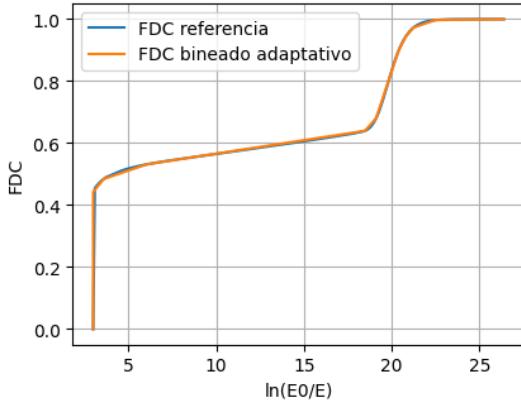
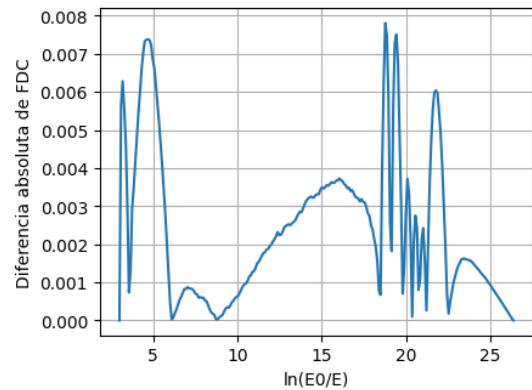


Figura A.10: Distribución de letargía aproximada con 10 bines. Se observa que el método mejora la aproximación de la distribución en la región de termalización y en la región epitérmica.



(a) FDC original vs. FDC con 10 bines.



(b) Diferencia absoluta entre FDCs.

Figura A.11: Evaluación del criterio de refinamiento para el caso de 10 bines. Se observa en la diferencia absoluta entre FDCs los valores de mayor discrepancia, que indican que los próximos bins deben ser ubicados en esas posiciones.

Caso con 25 bines

En la Figura A.12 se muestra la distribución de letargía aproximada con 25 bines. En este caso, el histograma logra capturar la delta en 1 MeV y mejora significativamente la aproximación de la distribución en las regiones epitérmica y térmica, salvo para letargías mayores a 22. En la Figura A.13a se presentan las FDCs de la distribución original de referencia y la obtenida con 25 bines, mientras que en la Figura A.13b se muestra la diferencia absoluta entre ambas FDCs, que sirve como criterio para determinar la ubicación del nuevo bin en la siguiente iteración.

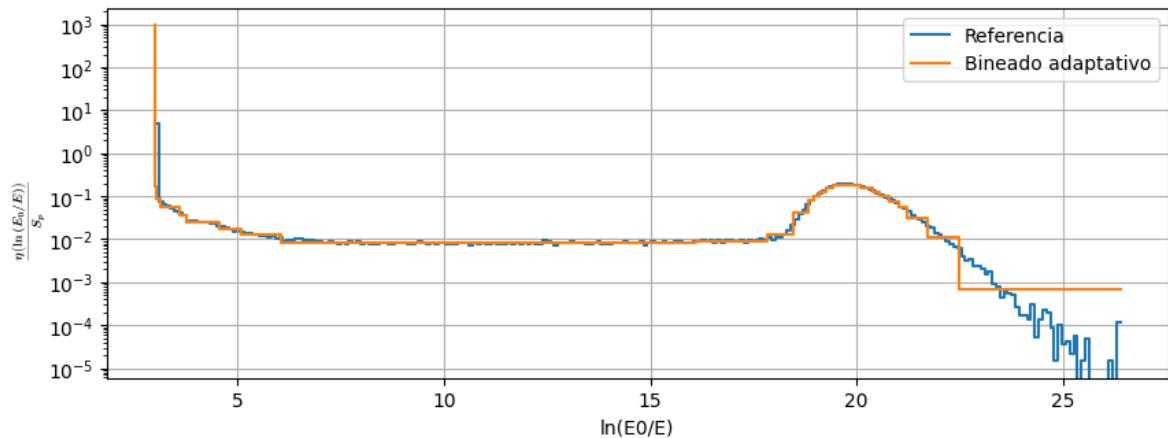
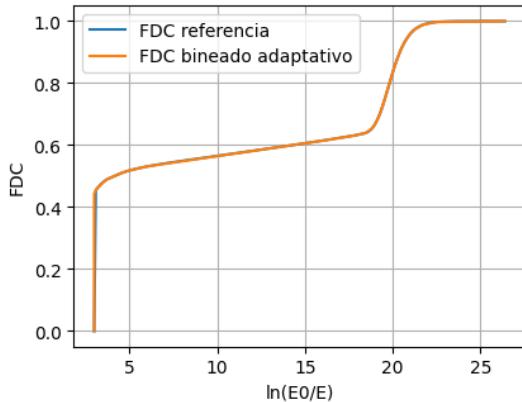
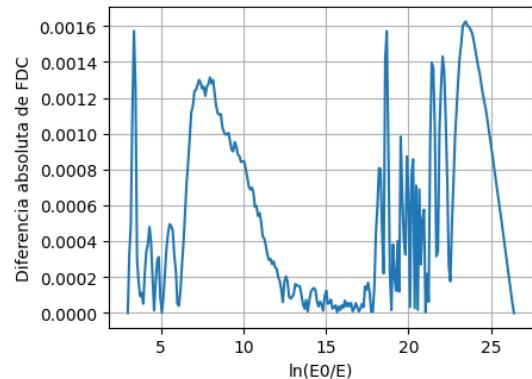


Figura A.12: Distribución de letargía aproximada con 25 bines. Se observa que el método logra capturar la distribución de letargía, salvo en la región de letargía mayor a 22, debido a la poca intensidad que tiene la distribución en esa región.



(a) FDC original vs. FDC con 25 bines.



(b) Diferencia absoluta entre FDCs.

Figura A.13: Evaluación del criterio de refinamiento para el caso de 25 bines. Se observa en la diferencia absoluta entre FDCs los valores de mayor discrepancia, que indican que los próximos bins deben ser ubicados en esas posiciones.

Caso con 100 bines

En la Figura A.14 se muestra la distribución de letargía aproximada con 100 bines. En este caso, el histograma logra capturar con precisión la distribución de letargía, salvo en la región de letargía mayor a 22, debido a la poca intensidad que tiene la distribución en esa región. Sin embargo se observa una mejoría con respecto al caso de 25 bines, donde la diferencia absoluta llega hasta 0.0016 y en el caso de 100 bines llega hasta 0.00012, aproximadamente un orden de magnitud menos. En la Figura A.15a se presentan las FDCs de la distribución original de referencia y la obtenida con 100 bines, mientras que en la Figura A.15b se muestra la diferencia absoluta entre ambas FDCs, que sirve como criterio para determinar la ubicación del nuevo bin en la siguiente iteración.

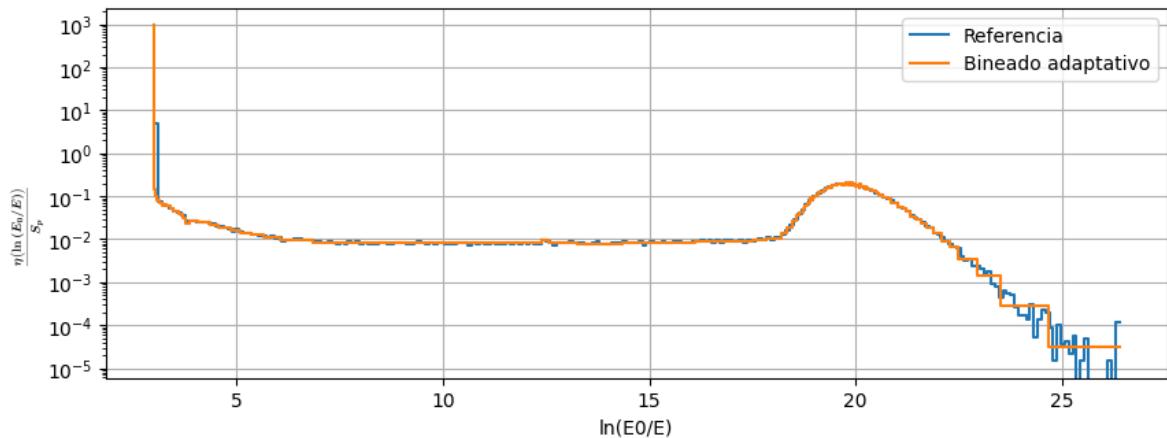
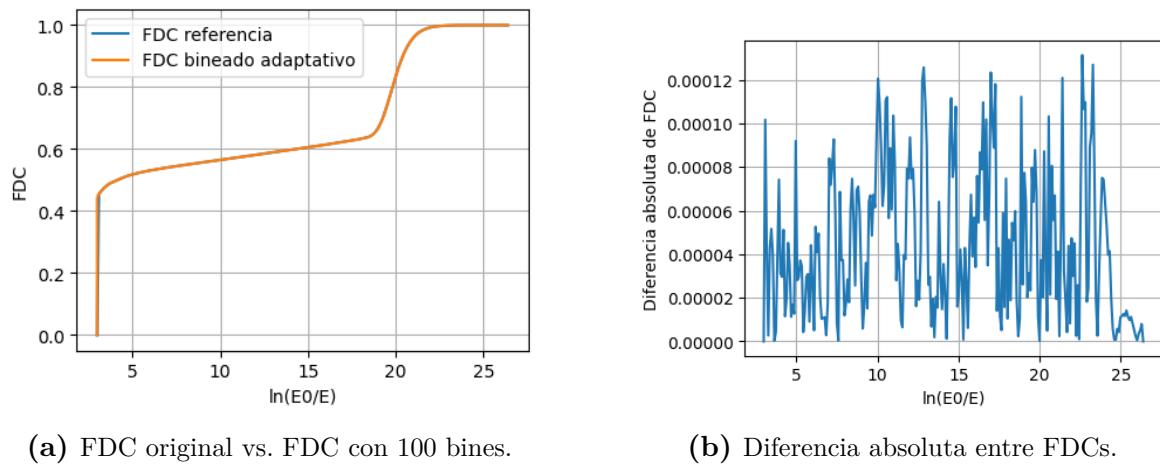


Figura A.14: Distribución de letargía aproximada con 100 bines. Se observa que el método logra capturar con precisión la distribución de letargía, salvo en la región de letargía mayor a 22, debido a la poca intensidad que tiene la distribución en esa región.



(a) FDC original vs. FDC con 100 bines.

(b) Diferencia absoluta entre FDCs.

Figura A.15: Evaluación del criterio de refinamiento para el caso de 100 bins.

Apéndice B

Implementación computacional del método de generación de fuentes Monte Carlo mediante histogramas multidimensionales

En este apéndice se presenta la implementación computacional del método de generación de fuentes Monte Carlo mediante histogramas multidimensionales. Se detallan los principales componentes, abarcando tanto el procesamiento en `Python` y `C/C++` como la integración con `OpenMC`:

- El procesamiento de un archivo `HDF5` en `Python`, incluyendo la construcción de la estructura `TreeNode` a partir de los datos de partículas.
- El algoritmo de *bineado* adaptativo implementado en `Python` y un ejemplo de cómo se genera el archivo `XML` resultante.
- La implementación en `C/C++` de la clase `HistogramSource` dentro de `OpenMC`, con foco en la función `fill_particle` y en la forma de utilizar esta fuente para una simulación.

El código fuente completo del proyecto se encuentra disponible en un repositorio público de *GitHub* [17] [18].

B.1. Procesamiento de un archivo `HDF5` en `Python`

Para extraer la información de un archivo `HDF5` generado por `OpenMC` y construir los histogramas multidimensionales, se emplea el módulo `histograms.py`. A continuación se describe el procedimiento principal:

1. Lectura de variables del espacio de fases (energía, posición y ángulo) y pesos estadísticos.

2. Construcción de un `DataFrame` de pandas que contenga:

$$\{\ln(E_0/E), x, y, \mu, \phi, \text{wgt}\}.$$

3. Ejecución recursiva de la función `build_node`, que genera un árbol de nodos `TreeNode`.

B.1.1. Estructura `TreeNode`

Cada nodo del árbol jerárquico almacena la distribución acumulada (CDF), los bordes de micro-bines, los bordes de macro-bines y los punteros a sus hijos. La clase en Python se define de la siguiente forma:

```

1  class TreeNode:
2      """Nodo elemental de un árbol jerárquico de histogramas multidimensionales.
3
4      Cada nodo contiene:
5          - **distribucion_acumulada** - Vector CDF (incluye un 0 inicial) para muestreo.
6          - **bordes_micro** - Bordes del histograma fino (bins "micro").
7          - **bordes_macro** - Bordes del histograma grueso (bins "macro").
8          - **hijos** - Lista de subnodos.
9
10     Atributos
11     -----
12         distribucion_acumulada : Optional[np.ndarray]
13             Vector de la distribución acumulada (CDF). None en hojas sin datos.
14         bordes_micro : Optional[np.ndarray]
15             Bordes de bins finos. None en hojas sin datos.
16         bordes_macro : Optional[np.ndarray]
17             Bordes de bins gruesos para particionar en hijos.
18             None si es última dimensión.
19         hijos : List[TreeNode]
20             Lista de nodos hijos (subárboles) en orden de bins_macro.
21         """
22
23     # -----
24     # Constructor: convertimos a `np.ndarray` sólo si los vectores existen.
25     # -----
26     def __init__(self,
27                  distribucion_acumulada: Optional[Sequence[float]],
28                  bordes_micro: Optional[Sequence[float]],
```

```
30     bordes_macro: Optional[Sequence[float]],
31 ) -> None:
32     # Inicializamos lista de hijos vacía
33     self.hijos: List[TreeNode] = []
34
35     # Convertimos secuencias a np.ndarray si existen, manteniendo dtype float
36     if distribucion_acumulada is not None:
37         self.distribucion_acumulada: np.ndarray = np.asarray(
38             distribucion_acumulada, dtype=float
39         )
40     else:
41         self.distribucion_acumulada = None
42
43     if bordes_micro is not None:
44         self.bordes_micro: np.ndarray = np.asarray(
45             bordes_micro, dtype=float
46         )
47     else:
48         self.bordes_micro = None
49
50     if bordes_macro is not None:
51         self.bordes_macro: np.ndarray = np.asarray(
52             bordes_macro, dtype=float
53         )
54     else:
55         self.bordes_macro = None
56
57     def add_child(self, hijo: "TreeNode") -> None:
58         """Añade un nodo hijo al final de la lista de hijos.
59
60         Args:
61             hijo (TreeNode): Nodo que se desea incorporar como subárbol.
62         """
63         self.hijos.append(hijo)
64
65     def to_xml(self) -> ET.Element:
66         """Serializa el nodo (y recursivamente sus hijos) a un elemento XML.
67
68         Cada nodo se representa como:
69         <node>
70             <cumul>...valores separados por comas o "None"</cumul>
71             <micro>...bordes_micro separados por comas o "None"</micro>
72             <macro>...bordes_macro separados por comas o "None"</macro>
73             <!-- subnodos aquí -->
74         </node>
75
```

Implementación computacional del método de generación de fuentes Monte Carlo mediante histogramas multidimensionales

```
76     Returns:  
77         ET.Element: Elemento <node> con sus etiquetas internas y subnodos.  
78     """  
79     nodo_elem = ET.Element("node")  
80  
81     # Convertimos cada vector a texto CSV; si es None, escribimos "None"  
82     etiquetas = ("cumul", "micro", "macro")  
83     vectores = (  
84         self.distribucion_acumulada,  
85         self.bordes_micro,  
86         self.bordes_macro,  
87     )  
88  
89     for etiqueta, vector in zip(etiquetas, vectores):  
90         subelem = ET.SubElement(nodo_elem, etiqueta)  
91         if vector is None:  
92             subelem.text = "None"  
93         else:  
94             # .tolist() asegura que sea iterable de Python puro  
95             subelem.text = ",".join(map(str, vector.tolist()))  
96  
97     # Añadimos los hijos de forma recursiva  
98     for hijo in self.hijos:  
99         nodo_elem.append(hijo.to_xml())  
100  
101     return nodo_elem
```

En esta estructura:

- `distribucion_acumulada` contiene la función de distribución acumulada (CDF) normalizada, incluyendo un cero inicial. Su longitud es igual al número de micro-bines más uno.
- `bordes_micro` define los bordes del histograma fino (micro-bines), utilizados para el muestreo de la variable en el nodo actual.
- `bordes_macro` define los bordes del histograma grueso (macro-bines), que partitionan el dominio y determinan los subnodos (hijos).
- `hijos` es la lista de subnodos jerárquicos, con una cantidad dada por

$$|\text{hijos}| = |\text{bordes_macro}| - 1,$$

ya que cada intervalo definido por `bordes_macro` representa una rama independiente del árbol.

La clase incluye tres métodos principales:

- `__init__`: inicializa un nodo, convirtiendo los vectores de entrada en arreglos de tipo numpy si están definidos, y creando una lista vacía de hijos.
- `add_child`: incorpora dinámicamente un subnodo al árbol, agregándolo al final de la lista hijos.
- `to_xml`: serializa el nodo actual y sus hijos a un elemento XML, escribiendo los vectores como cadenas separadas por comas y anidando recursivamente los subnodos. Este método permite almacenar la estructura completa del árbol jerárquico en un único archivo XML.

B.1.2. Construcción recursiva del árbol

La función principal que construye este árbol se denomina `build_node`. Esta función recibe el `DataFrame` con la información del archivo de partículas y la configuración de procesamiento. Su código es:

```

1 def build_node(
2     df: pd.DataFrame,
3     columns: Sequence[str],
4     micro_bins: Sequence[int],
5     macro_bins: Sequence[int],
6     micro_initial_bins: Sequence[Optional[int]],
7     macro_initial_bins: Sequence[Optional[int]],
8     micro_binning: str,
9     macro_binning: str,
10    user_edges: Sequence[Optional[List[float]]],
11 ) -> TreeNode:
12     """
13     Construye recursivamente un TreeNode a partir de un DataFrame, discretizando
14     sucesivamente según la lista de `columns` en dimensiones jerárquicas.
15
16     Cada nivel crea:
17         - macro_edges: bordes de macro-bins (si hay más dimensiones por procesar),
18         - micro_edges + cumul: bordes de micro-bins y su CDF correspondiente.
19
20     Parámetros
21     -----
22     df : pd.DataFrame
23         Subconjunto de tracks (cada fila es un evento), que debe contener:
24             - Columnas numéricas en `columns`.
25             - Columna "wgt" con pesos para cada fila.
26     columns : Sequence[str]
```

Implementación computacional del método de generación de fuentes Monte Carlo mediante histogramas multidimensionales

```
27     Lista de nombres de columnas que se discretizarán en orden jerárquico.
28     micro_bins : Sequence[int]
29         Número de micro-bins a generar para cada dimensión (paralelo a `columns`).
30     macro_bins : Sequence[int]
31         Número de macro-bins a generar para cada dimensión (paralelo a `columns`).
32     micro_initial_bins : Sequence[Optional[int]]
33         Parámetros initial_bins para binning adaptativo en micro-bins por dimensión.
34         Debe tener la misma longitud que `columns`.
35     macro_initial_bins : Sequence[Optional[int]]
36         Parámetros initial_bins para binning adaptativo en macro-bins por dimensión.
37         Debe tener la misma longitud que `columns`.
38     micro_binning : str
39         Estrategia de micro-binning: 'equal_bins' o 'adaptive'.
40     macro_binning : str
41         Estrategia de macro-binning: 'equal_bins' o 'adaptive'.
42     user_edges : Sequence[Optional[List[float]]]
43         Lista de listas de bordes de usuario para cada dimensión.
44         Cada elemento puede ser None (no hay bordes forzados) o una lista de floats.
45
46     Retorna
47     -----
48     TreeNode
49         Nodo raíz que contiene:
50             - cumul: CDF de la primera dimensión (o None si no aplica).
51             - micro_edges: bordes de micro-bins en la primera dimensión.
52             - macro_edges: bordes de macro-bins en la primera dimensión.
53         Y sus hijos correspondientes para las siguientes dimensiones.
54     """
55     # -----
56     # 1. Caso base: si no hay eventos, devolvemos un nodo "vacío" sin bordes.
57     # -----
58     if df.empty:
59         # TreeNode acepta (cumul, micro_edges, macro_edges).
60         # Pasamos None para indicar vacío.
61         return TreeNode(None, None, None)
62
63     # Nombre de la columna actual sobre la que estamos trabajando
64     col = columns[0]
65
66     # -----
67     # 2. Manejo de la "especialidad": todos los eventos en esta dimensión
68     # son idénticos
69     # -----
70     # Si todos los valores en df[col] son iguales, no tiene sentido calcular
71     # múltiples bins.
72     # Definimos micro_edges triviales y, si hay más dimensiones, generamos
```

```
73     # macro_edges minimal.
74     if df[col].min() == df[col].max():
75         valor_constante = float(df[col].min())
76
77     # Si es la última dimensión, no necesitamos macro_edges
78     if len(columns) == 1:
79         macro_edges: Optional[np.ndarray] = None
80     else:
81         # Para asegurar que haya al menos dos extremos y no crashear
82         # np.digitize, generamos un rango [valor-1, valor+1] (o cualquier
83         # delta pequeño).
84         macro_edges = np.array(
85             [valor_constante - 1.0, valor_constante + 1.0]
86         )
87
88     # micro_edges queda en un único punto (el valor constante)
89     micro_edges = np.array([valor_constante], dtype=float)
90     # La CDF acumulada es trivial: toda la probabilidad en ese único punto
91     cumul = np.array([1.0], dtype=float)
92
93 else:
94     # -----
95     # 3. Cálculo de macro_edges (si corresponde) en esta dimensión
96     # -----
97     if len(columns) == 1:
98         # Última dimensión: no necesitamos macro_edges
99         macro_edges = None
100    else:
101        # Llamamos a la función refactorizada `determine_macro_edges`
102        macro_edges = determine_macro_edges(
103            df,
104            col,
105            macro_bins[0],
106            macro_binning,
107            user_edges[0],
108            initial_bins=macro_initial_bins[0],
109        )
110        if macro_edges is None:
111            print("macro_edges is None")
112
113    # -----
114    # 4. Cálculo de micro_edges y su CDF (cumul) en esta dimensión
115    # -----
116    micro_edges, cumul = compute_micro_edges_and_cumul(
117        df,
118        col,
```

```

119         micro_bins[0],
120         micro_binning,
121         macro_edges,
122         initial_bins=macro_initial_bins[0],
123     )
124
125     # Creamos el nodo actual con los resultados en esta dimensión
126     node = TreeNode(cumul, micro_edges, macro_edges)
127
128     # -----
129     # 5. Caso recursivo: si hay más columnas, creamos hijos para cada macro-bin
130     # -----
131     if len(columns) > 1:
132         # Si no hay macro_edges (por ser última dimensión), no hay recursión
133         if macro_edges is None:
134             return node
135
136         # 5.1 Calculamos el índice de macro-bin para cada fila en df
137         # np.digitize devuelve índices de bin (1-based) para cada valor en df[col].
138         # Restamos 1 para pasar a 0-based.
139         idx = np.digitize(df[col].to_numpy(), bins=macro_edges) - 1
140
141         total_macro_bins = (
142             len(macro_edges) - 1
143         ) # Cantidad de intervalos macrobins
144
145         for i in range(total_macro_bins):
146             # -----
147             # 5.2 Construcción de la máscara para extraer sub-DataFrame en el bin i
148             #
149             # Truco de KDSource: en el penúltimo bin (i == total_macro_bins - 2),
150             # incluimos también los que quedaron en el último bin (i+1)
151             # para abarcar el rango abierto derecho.
152             # if total_macro_bins > 1 and i == total_macro_bins - 2:
153             #     mask = (idx == i) | (idx == i + 1)
154             # else:
155             #     mask = idx == i
156
157             mask = (
158                 (idx == i) | ((i == len(macro_edges) - 2) & (idx == i + 1))
159                 if len(macro_edges) > 2
160                 else (idx == i)
161             )
162
163             # df filtrado de eventos cuya coordenada col cae en el macro-bin i
164             child_df = df[mask]

```

```
165
166      # -----
167      # 5.3 Llamada recursiva: descartamos la primera columna y avanzamos
168      # -----
169      child_node = build_node(
170          child_df,
171          columns[1:],    # Quitamos la primera dimension
172          micro_bins[1:], # Micro-bins de las dimensiones restantes
173          macro_bins[1:], # Macro-bins de las dimensiones restantes
174          micro_initial_bins[
175              1:
176          ],   # Parámetros initial_bins para micro-bins
177          macro_initial_bins[
178              1:
179          ],   # Parámetros initial_bins para macro-bins
180          micro_binning, # Misma estrategia para todas las dimensiones
181          macro_binning, # Misma estrategia para todas las dimensiones
182          user_edges[1:], # Bordes de usuario para dimensiones restantes
183      )
184      # Agregamos el nodo hijo a la lista de hijos del nodo actual
185      node.add_child(child_node)
186
187      return node
```

La función `build_node` implementa el método desarrollado en el trabajo. Su objetivo es generar un árbol de tipo `TreeNode` a partir de un conjunto de partículas representado como un `DataFrame`. La estructura resultante representa de forma recursiva la distribución multidimensional de las variables.

Cada nodo del árbol corresponde a una variable del espacio de fases, siguiendo el orden dado en `columns`. En cada nivel, el conjunto de eventos se divide primero mediante un histograma grueso (`macro_bins`), generando subregiones que determinan los nodos hijos. Luego, dentro de cada subregión, se estima la distribución acumulada (CDF) sobre una grilla de histogramas finos (`micro_bins`).

En particular:

- Si todos los eventos presentan el mismo valor en una dimensión, se utiliza una discretización trivial y se evita la creación innecesaria de bins.
- Si no hay más dimensiones por procesar, el nodo creado se considera una hoja del árbol.
- Si existen más dimensiones, se invoca recursivamente `build_node` sobre cada subconjunto definido por los macro-bins, construyendo los hijos correspondientes.

Esta estructura es la que luego se serializa en formato XML mediante el método `to_xml` para el muestreo de partículas en etapas posteriores.

B.2. *Bineado adaptativo y generación del XML*

En esta sección se detalla el algoritmo de *bineado adaptativo* utilizado tanto para los macro-bines como para los micro-bines. Se ejemplifica a continuación la función para obtener bordes adaptativos en una sola dimensión:

B.2.1. Función de *bineado adaptativo* en Python

```

1  def obtener_bordes_adaptativos(
2      datos: np.ndarray,
3      num_bins: int,
4      pesos: Optional[np.ndarray] = None,
5      *,
6      initial_bins: int = 6,
7      fine_bins: int = 50_000,
8      user_edges: Optional[Sequence[float]] = None,
9  ) -> np.ndarray:
10     """
11         Calcula bordes adaptativos para un histograma de `num_bins` usando el
12         criterio  $|\Delta CDF|_{\max}$ .
13
14         El procedimiento realiza los siguientes pasos:
15         1. Construye un arreglo inicial de `initial_bins + 1` bordes
16             equiespaciados entre el mínimo y máximo de `datos`.
17         2. Inserta, si se proporcionan, los `user_edges` dentro del rango
18             [mín, max], descartando los que queden fuera y eliminando duplicados.
19         3. Calcula una CDF de alta resolución (`fine_bins + 1` bordes) para tener
20             una referencia "casi continua" de la distribución de los datos.
21         4. Repite `extra_bordes = num_bins - initial_bins` veces:
22             a. Calcula la CDF actual usando los bordes gruesos.
23             b. Interpola la CDF gruesa en los puntos finos.
24             c. Encuentra el índice donde la diferencia absoluta
25                  $|CDF_fina - CDF_gruesa_interp|$  es máxima.
26             d. Inserta ese punto (de la división fina) como nuevo borde en el
27                 arreglo grueso.
28         5. Devuelve los `num_bins + 1` bordes finales, ordenados de menor a mayor.
29
30         Parámetros
31         -----
32         datos : np.ndarray
33             Vector unidimensional de datos de entrada.

```

```

34     num_bins : int
35         Número total de bines deseados (cantidad de intervalos = num_bins,
36         cantidad de bordes = num_bins + 1).
37     pesos : Optional[np.ndarray]
38         Arreglo de pesos asociado a cada dato (misma longitud que `datos`).
39         Si es None, se asume peso uniforme = 1 para cada punto.
40     initial_bins : int, opcional (por defecto = 6)
41         Número inicial de bines gruesos (antes de refinar).
42         Debe cumplir 1 < initial_bins < num_bins.
43     fine_bins : int, opcional (por defecto = 50000)
44         Resolución de la malla fina usada para estimar la CDF de referencia.
45         Entre más grande, más precisa la ubicación de los bordes óptimos, pero
46         aumenta el costo computacional.
47     user_edges : Optional[Sequence[float]], opcional
48         Lista de bordes que el usuario quiere insertar obligatoriamente. Estos
49         se agregan sólo si están dentro del rango [mín(datos), máx(datos)].

50
51     Retorna
52     -----
53     np.ndarray
54         Vector ordenado de longitud `num_bins + 1` con los bordes adaptativos
55         que garantizan minimizar el error máximo de la CDF en cada paso.
56     """
57     if num_bins <= initial_bins:
58         raise ValueError(
59             "El número de bines deseados 'num_bins' debe ser mayor que
60             'initial_bins'."
61         )
62
63     # 1. Determinamos los extremos de los datos
64     minimo = float(np.min(datos))
65     maximo = float(np.max(datos))
66
67     # 2. Construimos bordes equiespaciados iniciales (initial_bins + 1)
68     bordes = np.linspace(minimo, maximo, initial_bins + 1)
69
70     # 3. Insertamos bordes de usuario si corresponden al rango y eliminamos
71     # duplicados
72     if user_edges is not None:
73         # Filtrar sólo los user_edges dentro de [mínimo, maximo]
74         bordes_usuario_filtrados = [
75             b for b in user_edges if minimo <= b <= maximo
76         ]
77         if bordes_usuario_filtrados:
78             # Concatenamos, unificamos y ordenamos
79             bordes = np.sort(

```

```

80             np.unique(np.concatenate((bordes, bordes_usuario_filtrados)))
81         )
82
83     # 4. Calculamos CDF de alta resolución para referencia
84     bordes_finos = np.linspace(minimo, maximo, fine_bins + 1)
85     _, cdf_fina = calcular_cdf_ponderada(datos, bordes_finos, pesos)
86
87     # 5. Insertamos iterativamente los bordes que maximizan |ΔCDF|
88     extra_bordes = num_bins - initial_bins
89     for _ in range(extra_bordes):
90         # 5.a. CDF en bordes gruesos actuales
91         _, cdf_gruesa = calcular_cdf_ponderada(datos, bordes, pesos)
92
93         # 5.b. Interpolamos la CDF gruesa en los puntos finos
94         # Usamos np.interp: interpola cdf_gruesa(bordes) en coordenadas
95         # bordes_finos
96         cdf_gruesa_interp = np.interp(bordes_finos, bordes, cdf_gruesa)
97
98         # 5.c. Calculamos la diferencia absoluta con la CDF fina y buscamos el
99         # máximo
100        diferencias = np.abs(cdf_fina - cdf_gruesa_interp)
101        indice_max = int(np.argmax(diferencias))
102
103        # 5.d. Insertamos el borde fino correspondiente y reordenamos
104        nuevo_borde = bordes_finos[indice_max]
105        bordes = np.sort(np.append(bordes, nuevo_borde))
106
107    return bordes

```

Donde la función auxiliar `calcular_cdf_ponderada` es:

```

1  def calcular_cdf_ponderada(
2      datos: np.ndarray, bordes: np.ndarray, pesos: Optional[np.ndarray] = None
3  ) -> Tuple[np.ndarray, np.ndarray]:
4      """
5          Calcula la función de distribución acumulada (CDF) ponderada en puntos dados.
6
7          Esta función construye un histograma de `datos` utilizando los `bordes`
8          especificados y, opcionalmente, aplica un arreglo de `pesos` a cada dato.
9          A partir de las frecuencias, se obtiene la CDF normalizada (entre 0 y 1).
10
11      Parámetros
12      -----
13      datos : np.ndarray
14          Vector unidimensional de datos a discretizar.
15      bordes : np.ndarray

```

```

16     Arreglo de longitud  $N+1$  que define los bordes de los bins donde se
17     evaluará la CDF.
18     pesos : Optional[np.ndarray]
19         Arreglo de pesos, de la misma longitud que `datos`. Si es None, se
20         asume peso=1 para cada dato.
21
22     Retorna
23     -----
24     Tuple[np.ndarray, np.ndarray]
25         - bordes : np.ndarray
26             Copia de los bordes de entrada (sin modificar).
27         - cdf : np.ndarray
28             Vector de longitud  $N+1$  con los valores de la CDF normalizada
29             evaluada en cada borde.
30             Notar que se inserta un 0 al inicio para representar la CDF antes
31             del primer bin.
32     """
33     # Obtenemos los conteos ponderados por bin
34     conteos, _ = np.histogram(datos, bins=bordes, weights=pesos)
35
36     # Construimos la CDF acumulada y agregamos un 0 inicial para la
37     # interpolación
38     cdf = np.insert(np.cumsum(conteos), 0, 0.0)
39
40     # Normalizamos la CDF en [0,1], evitando división por cero si todos los
41     # conteos son cero
42     total = cdf[-1]
43     if total:
44         cdf = cdf / total
45
46     return bordes.copy(), cdf

```

B.2.2. Ejemplo de archivo XML generado

Una vez construido el árbol completo (TreeNode), se genera el archivo XML que define la fuente para OpenMC. El formato general del archivo source.xml es el siguiente:

```

1 <HistogramSource>
2   <J> <!-- Corriente escalar -->
3     1.2345e-3
4   </J>
5   <ParticleType> neutron </ParticleType>
6   <z> 15.0 </z>
7   <VariableOrder> ln(E0/E),x,y,mu,phi </VariableOrder>
8   <SurfaceGeometry> rectangular </SurfaceGeometry>

```

```

9   <!-- Si SurfaceGeometry == circular, incluir <Radius> R </Radius> -->
10  <!-- A continuación, se anidan los nodos <node> del árbol de histogramas -->
11  <node>
12    <cumul>0.000,0.200,0.450,0.700,0.900,1.000</cumul>
13    <micro> ... valores separados por comas ... </micro>
14    <macro> ... valores separados por comas ... </macro>
15    <!-- Subnodos -->
16    <node>
17      <cumul> ... </cumul>
18      <micro> ... </micro>
19      <macro> ... </macro>
20      <!-- Y así sucesivamente -->
21    </node>
22    <node> ... </node>
23    <!-- ... -->
24  </node>
25 </HistogramSource>

```

En este XML:

- <J> contiene la corriente escalar $J = \sum w_i/N$.
- <ParticleType> indica “neutron” o “photon”.
- <z> es la coordenada de la superficie de registro (en cm).
- <VariableOrder> es la lista CSV de variables procesadas, por ejemplo ln(E0/E),x,y,mu,phi.
- <SurfaceGeometry> es “rectangular” o “circular”. En el caso circular, se incluye <Radius>R</Radius> debido a que esta opción permite verificar que las partículas generadas aparecen dentro de un radio R dado. En caso de que aparezcan fuera de este radio se vuelve a sortear la posición hasta que aparecen dentro del radio.
- Cada etiqueta <node> contiene:
 - <cumul>: valores de la CDF en los bordes de micro-nivel.
 - <micro>: bordes de micro-bines separados por comas.
 - <macro>: bordes de macro-bines separados por comas.
 - Subnodos anidados para las dimensiones siguientes.

De este modo, el archivo XML contiene toda la parametrización jerárquica que describe la fuente.

B.3. Implementación de HistogramSource en C/C++ para OpenMC

Para integrar la fuente jerárquica en OpenMC, se definió la clase `HistogramSource` en C++. A continuación se muestran los fragmentos más relevantes.

B.3.1. Definición de HistogramSource en `source.h`

```

1  /*! \class HistogramSource
2  * \brief Fuente compuesta a partir de histogramas cargados desde XML.
3  */
4  class HistogramSource : public Source {
5  public:
6      explicit HistogramSource(pugi::xml_node node);
7      ~HistogramSource() override;
8
9      // Muestra un punto del espacio de fases usando el árbol de histogramas
10     SourceSite sample(uint64_t* seed) const override;
11
12 private:
13     TreeNode* cumul_micro_macro; // Raíz del árbol (definido en histograms.h)
14     HSHeader* header;           // Estructura con J, z0, var_order, etc.
15     double z0;                 // Coordenada z de la superficie
16     ParticleType particle;    // Tipo de partícula ("neutron" o "photon")
17     std::vector<std::string> variable_order; // Orden de variables
18     std::string surface_geometry;
19     double surface_radius;    // Si circular
20 };

```

El constructor lee el XML y construye el árbol:

```

1  HistogramSource::HistogramSource(pugi::xml_node node)
2  {
3      // 1. Obtener el path al archivo XML
4      std::string path = get_node_value(node, "HistogramSource", false, true);
5
6      // 2. Parsear el archivo XML
7      pugi::xml_document doc;
8      pugi::xml_parse_result result = doc.load_file(path.c_str());
9      if (!result) {
10          throw std::runtime_error("Error al cargar XML de HistogramSource: " +
11                                 std::string(result.description()));
12      }
13 
```

```

14     pugi::xml_node root = doc.child("HistogramSource");
15
16     // 3. Leer la corriente J
17     pugi::xml_node j_node = root.child("J");
18     if (j_node) {
19         strength_ = std::stod(j_node.child_value());
20     } else {
21         throw std::runtime_error("El nodo <J> no está presente en el archivo XML.");
22     }
23
24     // 4. Leer el tipo de partícula
25     std::string particle_name = root.child("ParticleType").child_value();
26     particle = string_to_particle(particle_name);
27
28     // 5. Leer z0
29     z0 = std::stod(root.child("z").child_value());
30
31     // 6. Leer orden de variables
32     std::string order_string = root.child("VariableOrder").child_value();
33     variable_order = split_string(order_string);
34
35     // 7. Leer geometría de la superficie
36     surface_geometry = root.child("SurfaceGeometry").child_value();
37
38     // 8. Leer radio de la superficie
39     if (surface_geometry == "circular") {
40         surface_radius = std::stod(root.child("Radius").child_value());
41     }
42
43     // 9. Construir el árbol recursivamente desde el nodo <node>
44     pugi::xml_node root_node = root.child("node");
45     if (!root_node) {
46         throw std::runtime_error(
47             "No se encontró el nodo raíz <node> del árbol jerárquico.");
48     }
49     cumul_micro_macro = parse_node(root_node);
50
51     // 10. Lleno el header
52     HSHeader* header_aux = (HSHeader*)malloc(sizeof(HSHeader));
53     if (!header_aux) {
54         fprintf(stderr, "Error: No se pudo asignar memoria para header.\n");
55         exit(EXIT_FAILURE);
56     }
57     header_aux->J = strength_;
58     if (particle == ParticleType::neutron) {
59         header_aux->pype = strdup("neutron");

```

```

60     }
61     header_aux->z = z0;
62     header_aux->nvars = variable_order.size();
63     header_aux->var_order = (char**)malloc(header_aux->nvars * sizeof(char*));
64     for (int i = 0; i < header_aux->nvars; ++i) {
65         header_aux->var_order[i] = (char*)malloc(variable_order[i].size() + 1);
66         std::strcpy(header_aux->var_order[i], variable_order[i].c_str());
67     }
68
69     if (surface_geometry == "circular") {
70         header_aux->surface_geometry = strdup("circular");
71         header_aux->R = surface_radius;
72     } else if (surface_geometry == "rectangular") {
73         header_aux->surface_geometry = strdup("rectangular");
74     } else {
75         fprintf(stderr, "Error: Geometría de superficie no válida.\n");
76         exit(EXIT_FAILURE);
77     }
78
79     header = header_aux;
80 }
```

Este constructor de la clase `HistogramSource` se encarga de inicializar completamente la fuente a partir de un archivo XML. En primer lugar, extrae parámetros como la corriente escalar J , el tipo de partícula, la posición de la superficie z_0 , el orden de las variables del espacio de fases y la geometría de la superficie. Posteriormente, llama a una función recursiva que interpreta el árbol de nodos `<node>` desde el XML y lo transforma en una estructura de tipo `TreeNode`, que permite muestrear partículas conforme a la distribución multidimensional almacenada.

Finalmente, se completa la estructura auxiliar `HSHeader`, que contiene la información relevante para ser utilizada por el código en tiempo de ejecución, como ser el orden de variables, el tipo de superficie y, si corresponde, su radio. Esta estructura facilita el acceso a los metadatos durante el muestreo sin necesidad de volver a consultar el archivo XML.

B.3.2. Función `fill_particle` y `traverse` (`histograms.c`)

La función encargada de recorrer el árbol y llenar la estructura `mcpl_particle_t` es `fill_particle`. El proceso es:

1. Generar, para cada nivel del árbol, un número aleatorio uniforme en $[0, 1)$.
2. Interpolan linealmente sobre la CDF (`cumul`) para obtener un valor continuo.

3. Determinar el índice del macro-bin con `find_interval` y avanzar al nodo hijo.
4. Repetir hasta la última dimensión y asignar los valores a `mcpl_particle_t`.

A continuación se muestra el código de `traverse` y `fill_particle`:

```

1  /**
2   * Genera una partícula muestreada recursivamente a partir del árbol de histogramas.
3   *
4   * Esta función recorre el árbol de histogramas (estructura TreeNode) desde el nodo
5   * raíz hasta una hoja, extrayendo en cada nivel un valor muestreado según la
6   * distribución almacenada.
7   *
8   * El arreglo `valores_particula` debe tener la longitud adecuada (igual a la
9   * profundidad del árbol) y ya debe estar reservado por la función que invoca a
10  * `recorrer_arbol`.
11  *
12  * En cada nivel:
13  * 1. Se genera un número aleatorio uniforme en [0,1].
14  * 2. Se interpola linealmente este número sobre la CDF (`nodo_actual->cumul`)
15  *     para obtener un valor continuo `valor_muestreado` en la escala "micro"
16  *     (`nodo_actual->micro`).
17  * 3. Se determina a qué "macro-grupo" pertenece `valor_muestreado` mediante
18  *     la función find_interval(valor_muestreado, nodo_actual->macro,
19  *     nodo_actual->num_macro).
20  * 4. Se avanza al hijo correspondiente (subnodo) y se almacena `valor_muestreado`
21  *     en la posición `índice_nivel` del arreglo `valores_particula`.
22  *
23  * Cuando se llega a un nodo hoja (num_children == 0), se realiza un último muestreo
24  * en ese nodo y se guarda el valor en la última posición de `valores_particula`.
25  *
26  * @param[in] nodo_raíz           Puntero al nodo raíz del árbol (tipo TreeNode*).
27  * @param[out] valores_particula Arreglo de tipo double[] donde se almacenan los
28  *                               valores muestreados.
29  *                               Debe tener prealojada la memoria con tamaño
30  *                               igual a la profundidad del árbol.
31  *
32  * @nota El árbol `TreeNode` debe cumplir que en cada nodo intermedio:
33  *       - `micro` y `cumul` sean arreglos de igual longitud `num_micro > 0`.
34  *       - `macro` sea un arreglo ordenado.
35  *       - `children` sea un arreglo de punteros a `TreeNode` con
36  *         `num_children == num_macro - 1`.
37  *
38  * @nota Si en algún nivel find_interval retorna -1 (valor fuera de rango),
39  *       se imprimirá un mensaje de error en stderr y la función continuará con la
40  *       siguiente iteración.
41  *       Se asume que las distribuciones están correctamente definidas y
42  *       normalizadas en [0,1].

```

```
42  */
43 void traverse(TreeNode *nodo_raíz, double valores_particula[])
44 {
45     // Índice para recorrer el arreglo de salida
46     int indice_nivel = 0;
47
48     // Nodo actual inicia en la raíz
49     TreeNode *nodo_actual = nodo_raíz;
50
51     // Iterar mientras haya hijos (nodo no sea hoja)
52     while (nodo_actual->num_children > 0)
53     {
54         // 1. Generar número aleatorio uniforme en [0,1)
55         double aleatorio = (double)rand() / RAND_MAX;
56
57         // 2. Interpolar linealmente sobre la CDF (cumul) usando el arreglo de
58         //    micro-bins para obtener el valor continuo en "micro".
59         double valor_muestreado = interpolacion_lineal(
60             aleatorio,
61             nodo_actual->micro,
62             nodo_actual->cumul,
63             nodo_actual->num_micro);
64
65         // 3. Encontrar el índice de la macro-banda donde cae el valor muestreado
66         int idx_macro = find_interval(
67             valor_muestreado,
68             nodo_actual->macro,
69             nodo_actual->num_macro);
70
71         if (idx_macro < 0)
72         {
73             // Si el valor muestreado está fuera de rango de las bandas "macro"
74             fprintf(stderr,
75                     "Error: valor muestreado %.6f fuera de rangos macro",
76                     valor_muestreado,
77                     indice_nivel);
78             // Se continúa para intentar en el siguiente nivel (aunque lógicamente
79             // este caso no debería ocurrir si las distribuciones están bien
80             // definidas).
81         }
82         else
83         {
84             // 4. Avanzar al hijo correspondiente y guardar el valor en el arreglo
85             // de la particula
86             nodo_actual = nodo_actual->children[idx_macro];
87         }
}
```

Implementación computacional del método de generación de fuentes Monte Carlo mediante histogramas multidimensionales

```
88         valores_particula[indice_nivel] = valor_muestreado;
89         indice_nivel++;
90     }
91
92
93     // Al llegar a un nodo hoja, realizar un último muestreo en ese nodo
94     double aleatorio_final = (double)rand() / RAND_MAX;
95     double valor_final = interpolacion_lineal(
96         aleatorio_final,
97         nodo_actual->micro,
98         nodo_actual->cumul,
99         nodo_actual->num_micro);
100    valores_particula[indice_nivel] = valor_final;
101 }
```

```
1 /**
2 * Rellena un mcpl_particle_t con valores muestreados del árbol de histogramas.
3 *
4 * Esta función genera una partícula a partir de un árbol jerárquico de histogramas
5 * (TreeNode) y del encabezado HSHeader, que define parámetros como la geometría de
6 * la superficie ("rectangular" o "circular"), el radio R (si corresponde) y la
7 * coordenada z de la fuente.
8 * El muestreo se realiza nivel por nivel: en cada nivel se extrae un valor de la
9 * distribución acumulada (CDF) mediante interpolación lineal. En el caso de
10 * geometría circular, se impone la restricción  $x^2 + y^2 \leq R^2$  al muestrear los dos
11 * primeros grados de libertad espaciales.
12 *
13 * @param[out] particula_mcpl Puntero al struct mcpl_particle_t que se va a
14 * rellenar.
15 * @param[in] arbol_histograma Puntero al nodo raíz del árbol de histogramas
16 * (TreeNode).
17 * @param[in] encabezado_hs Puntero a HSHeader con parámetros de la fuente:
18 * - nvars: número de variables a muestrear en cada
19 * partícula.
20 * - surface_geometry: "rectangular" o "circular".
21 * - R: radio máximo permitido (solo si
22 * surface_geometry == "circular").
23 * - z: coordenada z fija de la superficie fuente.
24 *
25 * Notar:
26 * - Se reserva dinámicamente un arreglo intermedio de tamaño encabezado_hs->nvars
27 * para almacenar los valores muestreados en cada nivel del árbol. Al final de la
28 * función, este arreglo se libera.
29 * - Si ocurre cualquier error (memoria, geometría no soportada, o no poder
30 * muestrear dentro del círculo tras varios intentos), la función termina el
31 * programa con exit(EXIT_FAILURE).
```

```
32 */  
33 void fill_particle(mcpl_particle_t *particula_mcpl, TreeNode *arbol_histograma,  
34 HSHeader *encabezado_hs)  
35 {  
36     // Verificar parámetros obligatorios  
37     if (particula_mcpl == NULL || arbol_histograma == NULL || encabezado_hs == NULL)  
38     {  
39         fprintf(stderr, "Error en fill_particle: puntero nulo.\n");  
40         exit(EXIT_FAILURE);  
41     }  
42  
43     // Reservar espacio para almacenar los valores muestreados de la partícula  
44     double *valores_particula = (double *)malloc(encabezado_hs->nvars * sizeof(double));  
45     if (valores_particula == NULL)  
46     {  
47         fprintf(stderr, "Error en fill_particle: no se pudo asignar memoria.\n");  
48         exit(EXIT_FAILURE);  
49     }  
50  
51     // ----- Muestreo según la geometría definida en el encabezado -----  
52     if (strcmp(encabezado_hs->surface_geometry, "rectangular") == 0)  
53     {  
54         // Geometría rectangular: muestreamos sin restricción adicional  
55         traverse(arbol_histograma, valores_particula);  
56     }  
57     else if (strcmp(encabezado_hs->surface_geometry, "circular") == 0)  
58     {  
59         // Geometría circular: muestrear con restricción  $x^2 + y^2 \leq R^2$   
60         const int MAX_INTENTOS = 20;  
61         int intento;  
62         int exito = 1; // 0 = dentro del círculo, 1 = fuera o sin éxito  
63  
64         for (intento = 0; intento < MAX_INTENTOS; intento++)  
65         {  
66             exito = traverse_circular(arbol_histograma, valores_particula, encabezado_hs->R);  
67             if (exito == 0)  
68             {  
69                 // Se encontró un par (x,y) válido dentro del círculo  
70                 break;  
71             }  
72         }  
73         if (exito != 0)  
74         {  
75             fprintf(stderr,  
76                     "Error en fill_particle: no se generó una partícula válida  
77                     dentro del círculo después de %d intentos.",
```

Implementación computacional del método de generación de fuentes Monte Carlo mediante histogramas multidimensionales

```
78             MAX_INTENTOS);
79             free(valores_particula);
80             exit(EXIT_FAILURE);
81         }
82     }
83     else
84     {
85         // Geometría no soportada
86         fprintf(stderr,
87             "Error en fill_particle: geometría de superficie '%s' no soportada.
88             Use rectangular o circular.",
89             encabezado_hs->surface_geometry);
90         free(valores_particula);
91         exit(EXIT_FAILURE);
92     }
93
94     // ----- Conversión de valores muestreados a campos de mcpl_particle_t -----
95     // Valores esperados en valores_particula:
96     // [0] → ln(E0/E)      (primera variable muestreada)
97     // [1] → coordenada x (segunda variable)
98     // [2] → coordenada y (tercera variable)
99     // [3] → mu = cos(theta) (cuarta variable)
100    // [4] → phi (quinta variable)
101    // (Queda por generalizar otros orden de variables)
102
103    // 1. Energía cinética:
104    //     Relación E = 20 * exp(-ln(E0/E)) = 20 / exp(ln(E0/E))
105    //     Simplificando, E = 20 * exp(-valores_particula[0]).
106    //     (La constante 20 corresponde a E0 en MeV; ajustar según convenga).
107    particula_mcpl->eKin = 20.0 * exp(-valores_particula[0]);
108
109    // 2. Polarización: se inicializa en cero (no se emplea en este muestreo).
110    particula_mcpl->polarisation[0] = 0.0;
111    particula_mcpl->polarisation[1] = 0.0;
112    particula_mcpl->polarisation[2] = 0.0;
113
114    // 3. Posición espacial:
115    //     - x = valores_particula[1]
116    //     - y = valores_particula[2]
117    //     - z = coordenada fija definida en encabezado_hs->z
118    particula_mcpl->position[0] = valores_particula[1];
119    particula_mcpl->position[1] = valores_particula[2];
120    particula_mcpl->position[2] = encabezado_hs->z;
121
122    // 4. Dirección del vuelo:
123    //     La cuarta variable muestreada es   = cos(theta).
```

```

124     // La quinta variable es (ángulo azimutal).
125     // Entonces:
126     // theta = acos()
127     // direction_x = sin(theta) * cos(phi)
128     // direction_y = sin(theta) * sin(phi)
129     // direction_z =
130 {
131     double mu = valores_particula[3];
132     double phi = valores_particula[4];
133     double theta = acos(mu); // Ángulo polar
134     double sin_theta = sin(theta);
135
136     particula_mcpl->direction[0] = sin_theta * cos(phi);
137     particula_mcpl->direction[1] = sin_theta * sin(phi);
138     particula_mcpl->direction[2] = mu;
139 }
140
141 // 5. Tiempo de emisión: se asigna un valor fijo.
142 particula_mcpl->time = 10.0;
143
144 // 6. Peso estadístico de la partícula
145 particula_mcpl->weight = 1.0;
146
147 // 7. PDG code (neutron)
148 // 2112 corresponde a un neutrón. Ajustar si se muestrean otras partículas.
149 particula_mcpl->pdgcode = 2112;
150
151 // 8. User flags (sin uso; se inicializa en cero)
152 particula_mcpl->userflags = 0;
153
154 // Liberar memoria intermedia
155 free(valores_particula);
156 }
```

Las funciones `traverse` y `fill_particle` constituyen el muestreo de partículas a partir del árbol jerárquico de histogramas. La primera realiza un recorrido recursivo por la estructura `TreeNode`, interpolando valores en cada nivel según la CDF asociada y seleccionando el subnodo correspondiente. Este proceso se repite hasta alcanzar una hoja del árbol, generando un vector con las variables muestreadas del espacio de fases.

Por su parte, `fill_particle` utiliza este vector para completar todos los campos relevantes de la estructura `mcpl_particle_t`. Esto incluye la conversión inversa de la letargia a energía, la asignación de posición y dirección de vuelo según el sistema de coordenadas definido, y la adaptación a la geometría de superficie (circular o rectangular).

B.3.3. Inicialización de la fuente *on-the-fly* desde la API de OpenMC

La clase `HistogramSource` puede integrarse directamente en una simulación de OpenMC mediante su API de Python, sin necesidad de modificar el archivo `settings.xml`. Para ello, basta con instanciar la fuente a partir del archivo XML generado en la etapa de procesamiento y asignarla al objeto `settings`:

```
1 import openmc
2
3 settings = openmc.Settings()
4 settings.source = openmc.HistogramSource(
5     path="source.xml"
6 )
```

De esta manera, OpenMC cargará automáticamente la fuente al inicio de la simulación, utilizando muestreo *on-the-fly* directamente desde el árbol jerárquico definido en el archivo.

B.4. Ejemplo de flujo completo

A continuación se resume el flujo completo que permite integrar el método desarrollado dentro de una simulación Monte Carlo con OpenMC. Se describe el paso a paso desde la obtención del archivo de partículas hasta su uso como fuente distribucional.

1. **Generación del archivo HDF5 por OpenMC:** se ejecuta una simulación y se registran las partículas que atraviesan una superficie en un archivo `.h5` (por ejemplo, `tracks.h5`).
2. **Procesamiento en Python:** utilizando el módulo `HistogramSource` de `kdh`, se lee el archivo y se genera el árbol jerárquico de histogramas multidimensionales:

```
1 import kdsource.histograms as kdh
2
3 hs = kdh.HistogramSource(
4     trackfile="tracks.h5",
5     particle_type="neutron",
6     z0=0.0,
7     Nparticles=1e10,
8     surface_geometry="circular",
9     R=5.0,
10    domain={"w": [0, 2]},
11 )
```

```

12
13     hs.configure_binning(
14         variable_order=["ln(E0/E)", "x", "y", "mu", "phi"],
15         micro_bins=[75, 18, 18, 15, 10],
16         macro_bins=[3, 7, 4, 4],
17         micro_initial_bins=[1, 1, 1, 1, 1],
18         macro_initial_bins=[1, 1, 1, 1],
19         micro_binning="adaptive",
20         macro_binning="adaptive",
21     )
22
23     hs.build_tree()
24     hs.write_xml("source.xml")

```

3. Simulación con OpenMC usando la fuente *on-the-fly*: la fuente puede integrarse directamente desde la API de Python:

```

1 import openmc
2
3 settings = openmc.Settings()
4 settings.source = openmc.HistogramSource(path="source.xml")
5 settings.batches = 100
6 settings.particles = 100_000
7
8 settings.export_to_xml()
9 openmc.run()

```

4. Generación de un archivo de partículas: en caso de que se desee pre-generar un archivo .mcpl para utilizarlo como fuente en otras herramientas, puede hacerse con:

```

1     hs.generate_mcpl(
2         n_particles=1e7,
3         write_path="trackfile_generated.mcpl",
4         overwrite=True,
5     )

```


Bibliografía

- [1] McStas User Manual, 2020. Último acceso: junio de 2025. [2](#)
- [2] Schmidt, N., Abbate, O. I., Prieto, Z. M., Robledo, J. I., Márquez, J. I., Márquez, A. A., *et al.* Kdsource, a tool for the generation of monte carlo particle sources using kernel density estimation. *Annals of Nuclear Energy*, **177**, 109309, 2022. [2](#), [5](#)
- [3] Fairhurst Agosta, R. Cálculo neutrónico detallado de haces y guías de neutrones del reactor RA-10. Proyecto integrador de ingeniería nuclear, Instituto Balseiro, Centro Atómico Bariloche, 2017. [2](#), [5](#), [15](#)
- [4] Centro Atómico Bariloche. Centro atómico bariloche. <https://www.argentina.gob.ar/cnea/cab>. Último acceso: mayo de 2025. [5](#)
- [5] Ayala, J. E. Implementación de una línea de cálculo basada en el código TRIPOLI a problemas de blindaje del reactor RA-10. Proyecto integrador de ingeniería nuclear, Instituto Balseiro, Centro Atómico Bariloche, 2019. [5](#)
- [6] Abbate, O. I. Cálculo de blindajes mediante fuentes de distribución en McStas y Tripoli. Proyecto integrador de ingeniería nuclear, Instituto Balseiro, Centro Atómico Bariloche, 2020. [5](#)
- [7] Comisión Nacional de Energía Atómica. Comisión nacional de energía atómica. <https://www.cnea.gob.ar/>. Último acceso: mayo de 2025. [5](#)
- [8] Instituto Balseiro. Instituto balseiro. <https://www.ib.edu.ar/>. Último acceso: mayo de 2025. [5](#)
- [9] Abbate, O. I. KDSOURCE: Desarrollo de una herramienta computacional para el cálculo de blindajes. Tesis de maestría, Instituto Balseiro, Centro Atómico Bariloche, 2021. [5](#)
- [10] Fox, F. Optimización de algoritmos de estimación de densidades para el cálculo de haces de neutrones y fotones. Proyecto integrador de ingeniería nuclear, Instituto Balseiro, Centro Atómico Bariloche, 2022.

- [11] Giménez, M. A. Implementación de muestreo on-the-fly en el código KDSource. Proyecto integrador de ingeniería nuclear, Instituto Balseiro, Centro Atómico Bariloche, 2024. 5
- [12] OpenMC: Monte carlo code for nuclear simulation. <https://openmc.org/>, 2025. Último acceso: mayo de 2025. 5, 7
- [13] Hdf5®: The hierarchical data format. <https://docs.hdfgroup.org/archive/support/HDF5/doc/index.html>, 2025. Último acceso: mayo de 2025. 7, 8
- [14] MCPL: Monte carlo particle lists. <https://mctools.github.io/mcpl/>, 2025. Último acceso: mayo de 2025. 7, 8, 9
- [15] Documentación oficial de KDSource. <https://kdsource.readthedocs.io/en/latest/index.html>, 2024. Último acceso: mayo de 2025. 8
- [16] Extensible markup language (xml) 1.0 (fifth edition). <https://www.w3.org/TR/xml/>, 2008. Último acceso: mayo de 2025. 8
- [17] Ovando, L. Repositorio del proyecto de tesis - kdsource- generación de fuentes monte carlo mediante histogramas multidimensionales. <https://github.com/ovando912/KDSource>, 2025. Último acceso: junio de 2025. 18, 69
- [18] Ovando, L. Repositorio del proyecto de tesis - openmc- generación de fuentes monte carlo mediante histogramas multidimensionales. <https://github.com/ovando912/OpenMC>, 2025. Último acceso: junio de 2025. 18, 69
- [19] Schmidt, N. S. Desarrollo de un obturador neutrónico para la realización de espectrometría por tiempo de vuelo en el reactor RA-6. Tesis de maestría en ingeniería, Instituto Balseiro, Centro Atómico Bariloche, 2021. 45, 46, 48, 51
- [20] Departamento de Neutrones, Centro Atómico Bariloche. Figura provista por el departamento de neutrones del centro atómico bariloche. 47
- [21] Mishra, K. K., Hawari, A. I., Gillette, V. H. Design and performance of a thermal neutron imaging facility at the north carolina state university pulstar reactor. *IEEE Transactions on Nuclear Science*, **53** (6), 3904–3911, 2006. 53

Agradecimientos

A todos los que se lo merecen, por merecerlo.

