TESIS CARRERA DE MAESTRÍA EN INGENIERÍA

KDSOURCE: DESARROLLO DE UNA HERRAMIENTA COMPUTACIONAL PARA EL CÁLCULO DE BLINDAJES

Ing. Osiris Inti Abbate
Maestrando

Mgtr. Ariel Aníbal Márquez
Director

Dr. José Ignacio Márquez Damián
Co-director

Miembros del Jurado
Lic. Fernando Sánchez (Instituto Balseiro)
Ing. Daniel Hergenreder (INVAP S.E.)
Dr. Edmundo Lopasso (Instituto Balseiro)

Diciembre de 2021

Departamento de Física de Reactores y Radiaciones
Centro Atómico Bariloche

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

Inventario: 24441
9.06.22

Al bicho, que no se va más

Índice de símbolos

- P: Probabilidad.
- E: Energía.
- x, y, z: Variables de coordenadas espaciales, siendo z la dirección principal de propagación del haz, e y la dirección vertical.
- $\hat{\Omega}$: Dirección de propagación de una partícula.
- θ: Ángulo entre $\hat{\Omega}$ y la dirección de referencia (ej.: $\hat{z}$).
- $\mu = cos(\theta)$.
- ϕ: Ángulo entre el eje principal del plano normal a la dirección de referencia (ej.: $\hat{x}$) y la proyección de $\hat{\Omega}$ en dicho plano, siguiendo la regla de la mano derecha.
- H*(10): Dosis equivalente ambiental.
- Tracks: Propiedades de partículas (energía, posición y dirección) registradas al atravesar una superficie.
- Tally: Conteo de eventos en un volumen o superficie, o conjunto de éstos.
- Alocar: Reservar memoria para una variable (del inglés *allocate*).
- Samplear: Muestrear, obtener partículas de una fuente (del inglés *sample*).
- $[\cdot]$: Se utilizarán corchetes para indicar unidades. Corchetes vacíos significan que la magnitud es adimensional.

Índice de contenidos

Índice de figuras

Índice de tablas

Resumen

El presente proyecto consistió en el desarrollo de una herramienta computacional de modelado de fuentes distribucionales, denominada KDSource. La misma se orienta principalmente a aplicaciones en cálculos de blindajes y/o haces de partículas, y permite la comunicación con varios códigos Monte Carlo, con particular enfoque en McStas y TRIPOLI.

Se desarrolló un paquete de herramientas para el modelado general de fuentes de partículas por el método *Kernel Density Estimation*, método superador de la técnica de histogramas empleada anteriormente. Dichas fuentes se basan en listas de *tracks* capturadas en simulaciones Monte Carlo, y permiten la generación ilimitada de nuevas partículas respetando la distribución estimada, sin repeticiones. El paquete se compone de una librería en Python, a través de la cual se realiza el análisis y optimización de la fuente, y una librería en C, mediante la cual se generan nuevas partículas. Además cuenta con una aplicación de línea de comando y un conjunto de archivos auxiliares que facilitan la utilización de las fuentes. Los principales beneficios de la herramienta, en simulaciones Monte Carlo, son la reducción de varianza y el acople entre códigos.

Se demostró el funcionamiento y la utilidad de la herramienta en un cálculo de blindajes sobre un diseño conceptual de guía neutrónica, basado en el haz GF1 del reactor RA10. El cálculo incluyó un acople entre óptica y transporte de radiación, y permitió obtener mapas de dosis por neutrones, fotones *prompt*, fotones de fuente y fotones de activación en posiciones lejanas al núcleo.

Palabras clave: MONTE CARLO, BLINDAJES, KDE, MCSTAS, TRIPOLI

Abstract

The present project consisted in the development of a computational tool for distributional sources modelling, named KDSource. It is mainly oriented to shielding and/or particle beams calculations, and allows communication with several Monte Carlo codes, with special focus on McStas and TRIPOLI.

A tools package for general modelling of particle sources by Kernel Density Estimation was developed, being said method an overcomer of the previously used histograms technique. The sources are based on tracks lists captured in Monte Carlo simulations, and allow unlimited new particles generation respecting the estimated distribution, without repetitions. The package is composed of a Python library, through which analysis and optimization of sources is performed, and a C library, by means of which new particles are generated. It also includes a command line application and a set of auxiliary files which help sources utilization. The tool's main benefits, in Monte Carlo simulations, are variance reduction and coupling between codes.

The tool operation and utility was demonstrated in a shielding calculation on a conceptual neutron guide design, based on GF1 beam of RA10 reactor. The calculation included coupling between optics and radiation transport, and allowed to obtain dose maps caused by neutrons, prompt photons, source photones and activation photons at positions distant from the core.

Keywords: MONTE CARLO, SHIELDING, KDE, MCSTAS, TRIPOLI

Capítulo 1

Introducción

1.1. Objetivos y motivaciones

El cálculo de blindajes consiste en la obtención de valores de magnitudes dosimétricas mediante cálculos computacionales. Es de suma importancia en instalaciones con fuentes de radiaciones ionizantes, y particularmente en reactores nucleares. El objetivo final de los cálculos es determinar si parámetros como la tasa de dosis equivalente ambiental son adecuados para la presencia de personas, y/o qué clase de restricciones deberán ser tenidas en cuenta para la correcta protección radiológica del personal expuesto, de acuerdo a la normativa correspondiente. Normalmente es requerido el empleo de blindajes que separen la fuente de radiación de los recintos donde se pretende alojar trabajadores, cuyo material dependerá del tipo de partículas predominantes en la dosis ambiental.

Con respecto a los métodos computacionales empleados en el cálculo de blindajes, uno de los más utilizados, y al cual se restringe este proyecto, es el método Monte Carlo. Esto se se debe principalmente a que la gran anisotropía de los campos de radiación en medios absorbentes, más aún en haces de partículas, dificulta la aplicación de métodos determinísticos. El método Monte Carlo no introduce aproximaciones sobre la distribución angular del flujo, pero a costa de un alto costo computacional requerido para obtener resultados confiables en la zona de interés. La confiabilidad de un resultado depende, entre otras cosas, de su error estadístico, el cual disminuye con la cantidad de partículas empleadas para calcularlo.

En este sentido, emerge la siguiente dificultad: Al realizarse cálculos de blindajes, se emplea como fuente la radiación proveniente de una zona con altos niveles de flujo, y el punto de interés para el cálculo de dosis es la región donde se espera que ésta sea adecuada para las personas. Un alto factor de atenuación del flujo implica una baja probabilidad de que una partícula de fuente alcance la región de interés, lo cual significa que se debe producir una gran cantidad de partículas para obtener resultados

aceptables, lo cual a su vez se traduce en un alto costo computacional. Esto introduce la necesidad de técnicas de reducción de varianza, es decir métodos para propagar preferencialmente la radiación hacia la zona de interés, sin modificar la física del problema.

El objetivo del presente trabajo es el desarrollo de una herramienta de modelado de fuentes distribucionales, mediante la cual es posible implementar una técnica de reducción de varianza. Dicha metodología ya fue explorada en trabajos anteriores [1], [2], [3], con diversas aplicaciones. A continuación se describe en más detalle dicha técnica, las implementaciones anteriores, y los objetivos para la presente implementación.

1.2. Fuentes distribucionales en simulaciones Monte Carlo

La implementación de fuentes distribucionales es una técnica versátil que permite, entre otras cosas, la reducción de varianza. Los problemas de interés son entonces situaciones en las cuales no es posible alcanzar estadística suficiente en el detector de interés en una única corrida, pues el costo computacional sería demasiado alto. En cualquier simulación de este tipo, sin embargo, es posible determinar una superficie más cercana a la fuente en la cual sí es posible alcanzar buena estadística. Dicha situación se esquematiza en la Figura 1.1, y es en la que se basa la aplicación del método de fuentes distribucionales.

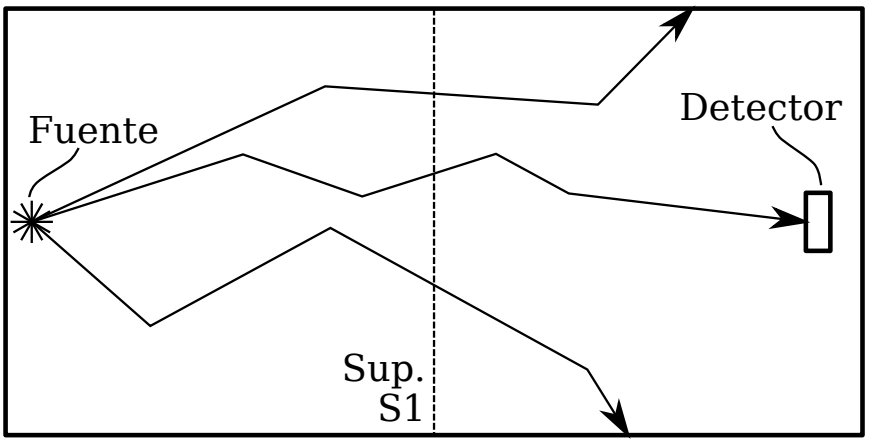


Figura 1.1: Esquema básico de una simulación por Monte Carlo. La superficie S1 permite la implementación de la técnica de fuentes distribucionales.

Continuando con el ejemplo de la Figura 1.1, el método consiste en registrar las partículas que atraviesan la superficie S1, obteniendo una lista de parámetros (energía, posición, dirección) con relativamente buena estadística, denominada lista de *tracks*, o simplemente lista de partículas. Esta es utilizada a continuación para estimar la distribución multidimensional de corriente en S1. Finalmente, se utiliza dicha distribución

estimada como fuente en una nueva simulación. La cantidad de partículas producidas en esta segunda etapa puede superar el tamaño de la lista original, mejorando la estadística en la región del detector y reduciendo la varianza del resultado. Todo este proceso se muestra en la Figura 1.2.

Una característica importante de las fuentes de radiación es la correlación entre variables, es decir, el hecho de que la distribución en función de cada variable tiene una dependencia con el valor de las otras. El modelado adecuado de dicha correlación

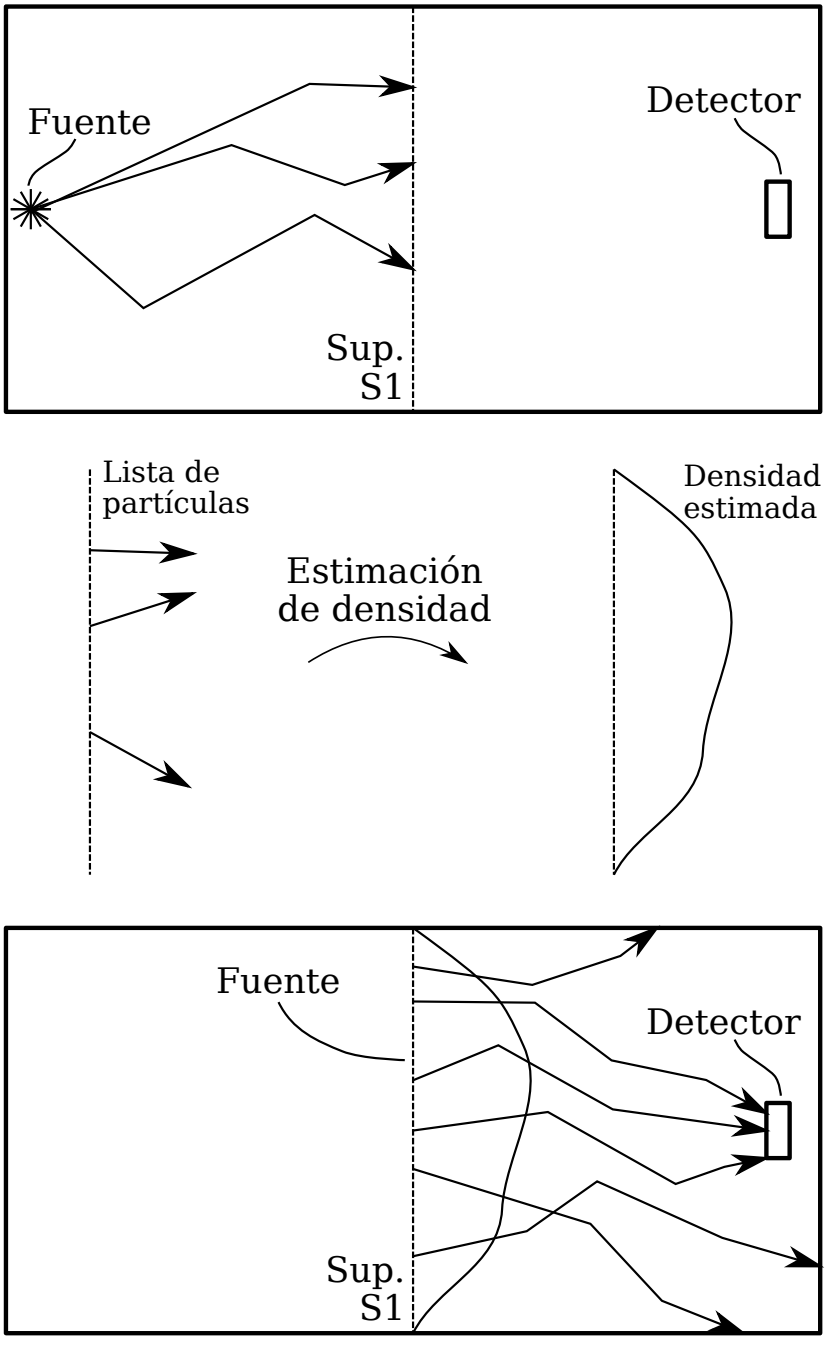


Figura 1.2: Esquema de la implementación del método de fuentes distribucionales. Se utiliza una lista de partículas registrada en una superficie intermedia para estimar su distribución, y utilizar esta última como fuente en una segunda simulación.

representa una dificultad notable en la creación de fuentes distribucionales, en parte debido a la estadística limitada de las listas de *tracks*. De hecho, es usual la representación de fuentes mediante distribuciones separadas para cada variable, a pesar de que esto representa una pérdida de información considerable. En los desarrollos del presente trabajo se evitará el uso de dichas técnicas, y, al contrario, se dará particular importancia a la adecuada representación de la correlación entre variables.

1.3. Desarrollos anteriores: fuentes de distribuciones mediante histogramas

En el Proyecto Integrador (PI) de mi carrera de grado [1], en continuación con los de Roberto Fairhurst [2] y Jonathan Ayala [3], se desarrollaron técnicas para la implementación de fuentes de distribuciones basadas en histogramas, para su uso en los códigos McStas y TRIPOLI-4 (ver 1.4). En el caso de Fairhurst, su uso estaba orientado a la propagación de neutrones por el interior de guías con McStas, para caracterizar las distribuciones a la salida de los haces de RA10. En el caso de Ayala, se introdujo el código TRIPOLI-4, y se demostró la utilidad de la cadena de cálculo en un cálculo de blindajes en un conducto de extracción también de RA10. En mis desarrollos se implementó la técnica de fuentes distribucionales sobre espejos de guías neutrónicas, permitiendo, mediante el acople de óptica neutrónica en McStas y transporte de radiación en TRIPOLI-4, la realización de cálculos de blindajes en la periferia de guías.

Cada utilización de una fuente de distribuciones consta de dos etapas, una de detección y otra de producción. En la etapa de detección, las partículas que atraviesan el plano de la fuente son clasificadas según un conjunto de variables adecuado según la geometría de la guía (por ejemplo, E, x, y, μ y ϕ). Dicha clasificación consiste en la creación de histogramas para cada variable, basados en una estructura de discretizaciones dada. Por el otro lado, la etapa de producción consiste en el muestreo de partículas basado en la distribución calculada, es decir la obtención de valores aleatorios de cada variable respetando las distribuciones presentes en los histogramas.

Con respecto a la correlación entre variables, la técnica elegida en estos desarrollos para capturar dicho fenómeno es el empleo de histogramas anidados, con esquemas como el que se muestra en la Figura 1.3. En ese caso, lo primero que se calcula es un histograma para la variable μ, y luego, para cada *bin* de μ, se calcula un histograma en E. A continuación, para cada bin de E se calcula un histograma en x, y así sucesivamente. Para mantener una estadística apropiada en todas las distribuciones, se define para cada variable una grilla gruesa (macro) y una fina (micro). Las grillas gruesas se utilizan para los histogramas "de correlación", es decir aquellos que preceden clasificaciones sobre otras variables, y las finas se utilizan para los histogramas finales.

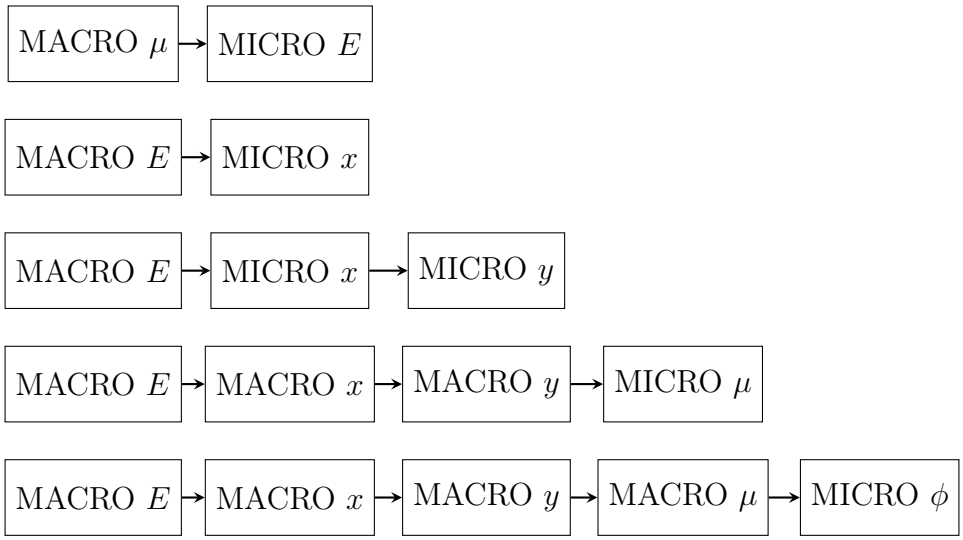


Figura 1.3: Esquema de la estructura de discretizaciones empleada para fuentes planas rectangulares.

Si bien la técnica de histogramas anidados permitió obtener resultados satisfactorios en cada uno de los trabajos mencionados, presenta algunos inconvenientes. El más significativo de ellos es la dificultad para generar y optimizar la estructura de discretizaciones, lo cual se realizó de forma manual. Otro problema es la poca versatilidad de las herramientas frente a diferentes geometrías de fuente, o tratamientos de variables deseados, ya que esquemas como el de la Figura 1.3 se encuentras fijados en la arquitectura del código. Por último, los histogramas, por definición, modelan las distribuciones como constantes a trozos. En el presente trabajo se buscó superar estas problemáticas mediante el empleo de técnicas más avanzadas de estimación de densidad.

1.4. Códigos de simulación Monte Carlo

En esta sección se presentan los dos principales códigos de simulación de radiación por Monte Carlo utilizados en este trabajo. No son los únicos para los cuales aplican los métodos y herramientas desarrolladas, pero sí en los que se enfocaron las simulaciones en las que se demuestra la utilidad de los mismos.

1.4.1. McStas

McStas [4–6] es un paquete de simulación de instrumentos neutrónicos por el método Monte Carlo, desarrollado en forma conjunta por DTU Physics y los institutos Laue Langevin (ILL), Paul Scherrer y Niels Bohr (NBI). Los modelos se construyen a través de un meta-lenguaje de alto nivel, en el cual se define una secuencia de componentes, incluyendo fuentes y de-

tectores de partículas así como instrumentos neutrónicos. El programa convierte dicho *input* en un código en ISO-C con el cual se ejecuta la simulación y se obtienen los resultados. Se trata de un *software* de código abierto, es decir que tanto el motor de cálculo como la implementación de los componentes incluidos, en su mayoría desarrollada en lenguaje C, puede descargarse libremente de la página oficial de McStas, mcstas.org.

Cada componente de McStas está definido en un archivo de texto, con una estructura que consta de bloques de código delimitados por comandos especiales, cuyos contenidos son en lenguaje C. Las transformaciones que sufre una partícula al atravesar cada componente están determinadas por los comandos allí definidos, con lo cual las posibilidades para instrumentos de óptica neutrónica son prácticamente ilimitadas. Sin embargo, esto hace que el código no sea útil para el cálculo de blindajes, ya que el modelado de materiales no está implementado.

Una de las grandes ventajas del carácter abierto de McStas es que pueden definirse nuevos componentes los cuales, en las simulaciones, resultan completamente equivalentes a los incluidos en el programa. Incluso es posible definir bibliotecas de desarrollo propio e incluirlas de forma simple en las definiciones de componentes. Estas características le dan al código una gran versatilidad, la cual resulta muy útil para el presente proyecto. En efecto, la implementación de fuentes de distribuciones en McStas, tanto para la detección como para la producción de partículas, se realizó mediante la definición de una biblioteca en lenguaje C y componentes como los descritos.

1.4.2. TRIPOLI-4

TRIPOLI-4 [7] es una herramienta de transporte de radiación por el método Monte Carlo, desarrollada por el CEA (*Commissariat à l'énergie atomique et aux énergies alternatives*) en Francia. Su motor de cálculo resuelve la ecuación de Boltzmann tridimensional para neutrones, fotones, electrones y positrones, permitiendo la resolución tanto de problemas de criticidad como de fuente fija (cálculo de blindajes). Su biblioteca oficial de secciones eficaces y datos nucleares es CEAV5.1.1, basada en las evaluaciones europeas JEFF-3.1.1. Entre sus características más destacables se incluyen: la implementación un método de reducción de varianza automático, la posibilidad de paralelización de los cálculos, y la posibilidad de incluir fuentes externas. La utilización de TRIPOLI-4 se encuentra motivada por el hecho de que, desde 2017, Argentina pertenece a NEA (Nuclear Energy Agency), con lo cual tiene acceso a la licencia de los códigos incluidos en su repositorio, entre los que se encuentra dicha herramienta.

En el presente proyecto se aprovechó la posibilidad de incorporar fuentes externas en TRIPOLI-4 para la incorporación de los desarrollos de fuentes de distribuciones. Dicha

funcionalidad se logra mediante la definición en lenguaje C de una función que al ser llamada provea los parámetros que definen una partícula. La misma debe compilarse como una biblioteca compartida antes de la simulación, pudiendo incluirse en el proceso código auxiliar o bibliotecas externas. Con respecto a la detección de partículas para su caracterización distribucional, se registraron listas de *tracks* en las zonas de interés, las cuales fueron procesadas posteriormente.

Capítulo 2

Formulación general del problema de estimación de densidad y muestreo

El problema matemático que se busca resolver al generar fuentes distribucionales es lo que se denomina "estimación de densidad", y muestreo (*sampling*) basado en dicha estimación. En adelante se denominarán ambos problemas en conjunto como "estimación de densidad y muestreo". En este capítulo se formulará de manera general dicho problema, y se analizarán los diferentes métodos existentes, con el objetivo de seleccionar el más apropiado para la aplicación a fuentes de radiación.

Sea $X = \{x^{(i)} \in \mathbb{R}^D\}_{1 \leq i \leq N}$, con $D \in \mathbb{N}$, un conjunto de N puntos muestreados de una distribución de probabilidad p desconocida. Se busca obtener una estimación $\tilde{p}$ lo más cercana a p posible, para una distancia dada (estimación de densidad). Además, se busca producir nuevas muestras $\tilde{X} = \{\tilde{x}^{(i)} \in \mathbb{R}^D\}_{1 \leq i \leq M}$, siguiendo la distribución $\tilde{p}$ (muestreo).

Para cuantificar la calidad de la estimación $\tilde{p}$ es necesario definir una distancia entre p y $\tilde{p}$. Además, resulta deseable contar con expresiones aproximadas de dicha distancia, en función de los datos X y $\tilde{p}$, dado que la distribución real p se supone desconocida. A continuación se presentan las distancias más usuales [8]:

- *Mean integrated square error* (MISE):

$$
\begin{aligned}
MISE(p, \tilde{p}) &= \int (p(x) - \tilde{p}(x))^2 dx && (2.1)\\
&= \int p(x)^2 dx - 2 \int p(x)\tilde{p}(x) dx + \int \tilde{p}(x)^2 dx \\
&\approx \frac{1}{N} \sum_{i=1}^{N} p(x^{(i)}) - 2\frac{1}{N} \sum_{i=1}^{N} \tilde{p}(x^{(i)}) + \frac{1}{M} \sum_{i=1}^{M} \tilde{p}(\tilde{x}^{(j)}) && (2.2)
\end{aligned}
$$

- Divergencia de Kullback-Leibler (KL):

$$\begin{aligned} KL(p,\tilde{p}) &= \mathbb{E}_{x\sim p(x)}\left(log\left(\frac{p(x)}{\tilde{p}(x)}\right)\right) \\ &= \int p(x) log\left(\frac{p(x)}{\tilde{p}(x)}\right) dx \\ &= \int p(x) log(p(x)) dx - \int p(x) log(\tilde{p}(x)) dx \\ &= CE(p,\tilde{p}) - CE(p,p) \end{aligned} \tag{2.3}$$

Donde se define la entropía cruzada $CE(p,\tilde{p})$ como:

$$CE(p,\tilde{p}) = -\int p(x) log(\tilde{p}(x)) dx \tag{2.4}$$

$$\approx -\frac{1}{N}\sum_{i=1}^{N} log(\tilde{p}(x^{(i)})) \tag{2.5}$$

$$\begin{aligned} &= -\frac{1}{N} log(\prod_{i=1}^{N} \tilde{p}(x^{(i)})) \\ &= -\frac{1}{N} log(P(X|\tilde{p})) \\ &= -\frac{1}{N} log\left(\frac{P(X)}{P(\tilde{p})} P(\tilde{p}|X)\right) \end{aligned}$$

Como se ve en la última expresión, $CE(p,\tilde{p})$ da la inversa de la log-probabilidad de $\tilde{p}$ dado X, según el teorema de Bayes. Además, dado que, al optimizar $\tilde{p}$, la distribución original p resulta constante, minimizar la Divergencia de Kullback-Leibler equivale a minimizar la entropía cruzada entre p y $\tilde{p}$. Por otra parte, mediante la expresión 2.5 para $CE(p,\tilde{p})$, esta se puede estimar sólo en función de $\tilde{p}$ y el conjunto de muestras X.

En general, el conjunto X tendrá una matriz de covarianza Σ, pero puede resultar útil aplicar una normalización de datos, lo cual consiste en preprocesar los datos para dar a dicha matriz una forma deseada. El método más simple es el escaleo de variables, es decir dividir cada componente (dimensión) de los datos por su desviación estándar σ_d. De este modo el conjunto normalizado tiene dispersión unitaria en todas sus variables, aunque se mantiene la correlación entre variables. La distribución p de los datos no normalizados puede obtenerse de la distribución p_n de los datos normalizados mediante:

$$p(x) = p(x_1, ..., x_D) = \frac{1}{\sigma_1 ... \sigma_D} p_n(\frac{x_1}{\sigma_1}, ..., \frac{x_D}{\sigma_D}) \tag{2.6}$$

El método de normalización más completo consiste en aplicar un cambio de variables tal que la matriz de covarianza sea unitaria, lo cual requiere una búsqueda de los

autovalores y autovectores de Σ.

A continuación se presentan y describen las técnicas de estimación de densidad y muestreo analizadas.

2.1. Histogramas

La técnica de histogramas consiste en proponer una $\tilde{p}$ constante por regiones, es decir:

$$\tilde{p}(x) = c_k, \ si \ x \in \Omega_k, \ k = 1, ..., K \tag{2.7}$$

Donde $\{\Omega_k\}_{1 \leq k \leq K}$ es un conjunto de regiones espaciales disjuntas que cubren todo el dominio de interés. Por normalización se debe tener $\sum_{k=1}^{K} c_k |\Omega_k| = 1$, con $|\Omega_k|$ la medida de Ω_k. Los c_k se obtienen de la minimización de $CE(p, \tilde{p})$, y tienen el valor esperado intuitivamente:

$$c_k = \frac{n_k}{N|\Omega_k|} \tag{2.8}$$

Donde n_k es la cantidad de datos de X dentro de Ω_k.

La principal dificultad de esta técnica es la determinación de las zonas Ω_k en las cuales dividir el dominio. Típicamente se toman como rectángulos D-dimensionales, reduciendo el problema a la elección de los anchos en cada dimensión de cada Ω_k (*bins*). Por otra parte, puede ser útil dividir la función $\tilde{p}(x)$ en condicionales $\tilde{p_1}(x_1)$, $\tilde{p_2}(x_2|x_1)$, ..., $\tilde{p}(x_D|x_1, ..., x_{D-1})$, ya que se reduce en problema D-dimensional a D problemas unidimensionales. Si se toman anchos de *bins* constantes, es posible utilizar las técnicas de optimización de ancho de banda para *Kernel Density Estimation*, descritas en la Subsección 2.3.2. Otra limitación intrínseca de los histogramas es la falta de suavidad de las distribuciones estimadas.

El muestreo con histogramas se realiza en 2 pasos:

1. Se elige la región Ω_k aleatoriamente, tomando como pesos estadísticos los c_k.

2. Se sortea un x uniformemente distribuido dentro de Ω_k.

Si se dividió $\tilde{p}(x)$ en D condicionales unidimensionales, este algoritmo se realiza secuencialmente sobre cada uno de ellos.

2.2. Modelos paramétricos

Es posible obtener $\tilde{p}$ ajustando una función con forma predeterminada. Es decir que, dada a una familia de funciones con un parámetro libre Φ, se toma aquella más cercana a p. Usualmente su determinación se basa en la minimización de $CE(p, \tilde{p})$.

El caso más sencillo (unimodal) es el de una única gaussiana, donde sus parámetros (media y varianza) pueden calcularse directamente de los datos. Desde luego las distribuciones que pueden ser representadas adecuadamente con este método son muy limitadas. El ajuste K-modal extiende esta metodología, volviéndola aplicable a distribuciones con varios "picos". En este caso la familia de funciones resulta:

$$\tilde{p}(x) = \sum_{k=1}^{K} c_k \mathcal{N}(\mu_k, \Sigma_k) \tag{2.9}$$

Donde $\mathcal{N}(\mu, \Sigma)$ es la distribución normal de media μ y matriz de covarianza Σ. La minimización de $CE(p, \tilde{p})$ suele ser iterativa, por ejemplo mediante gradiente descendente, siendo los parámetros a ajustar los $\{\mu_k, \Sigma_k\}_{1 \leq k \leq K}$.

El caso extremo en el cual $K = N$ deriva en el método *Kernel Density Estimation*.

2.3. *Kernel Density Estimation*

La técnica de *Kernel Density Estimation* (KDE) consiste en estimar la densidad mediante la siguiente fórmula [9]:

$$\tilde{p}(x) = \frac{1}{N|H|} \sum_{i=1}^{N} K(H^{-1}(x - x^{(i)})) \tag{2.10}$$

Donde:

- K es el *kernel*, el cual debe cumplir $\int K(u)du = 1$ y usualmente cumple $K(-u) = K(u)\ \forall u$ (simetría). Se dice que es un *kernel* producto si es de la forma $K(u) = k(u_1)...k(u_D)$, donde k es un *kernel* unidimensional.
- H es el ancho de banda (*bandwidth*). En general es una matriz, pero en el caso que sea diagonal puede representarse como un vector $H = (h_1, ..., h_D)$, y si todos los elementos de este último son iguales se reduce a un escalar $H = h$.

Para el *kernel* se definen los siguientes parámetros:

- Momentos: $\kappa_j(K) = \int u^j K(u)du$
- Orden: $\nu = min(\{j \in \mathbb{N} \ / \ \kappa_j \neq 0\})$
- Arpereza (*roughness*): $R(K) = \int K(u)^2 du$

En principio cada momento $\kappa_j(K)$ en un tensor de orden j, pero para un *kernel* producto se utilizan los $\kappa_j(k)$, los cuales son escalares.

El *kernel* no necesariamente debe cumplir $k(u) > 0\ \forall u$, pero de cumplirlo (no negatividad) y además ser simétrico, su orden será $\nu = 2$. La no negatividad asegura

que la densidad será no negativa en todo el dominio, pero, como se verá más adelante, los *kernel* de orden superior reducen el sesgo (*bias*) de $\tilde{p}$.

El error de $\tilde{p}(x)$ se compone de dos términos, el *bias* y la desviación estándar (raíz cuadrada de la varianza). Sus expresiones, para un *kernel* producto y H vector, son las siguientes:

$$\begin{aligned} Bias(\tilde{p}(x)) &= \mathbb{E}(\tilde{p}(x)) - p(x) \\ &= \frac{\kappa_\nu(k)}{\nu!} \sum_{d=1}^{D} h_d^\nu \frac{\partial^\nu p}{\partial x_d^\nu}(x) + O(h_1^\nu + ... + h_D^\nu) \end{aligned} \tag{2.11}$$

$$\begin{aligned} Var(\tilde{p}(x)) &= \frac{p(x)R(K)}{N|H|} + O(1/N) \\ &= \frac{p(x)R(k)^D}{Nh_1...h_D} + O(1/N) \end{aligned} \tag{2.12}$$

Por lo tanto el intervalo de confianza de $p(x)$, para $\nu = 2$, *kernel* producto y H vector, resulta:

$$\begin{aligned} p(x) &= \tilde{p}(x) - \frac{1}{2}\kappa_2(k) \sum_{d=1}^{D} h_d^2 \frac{\partial^2 p}{\partial x_d^2}(x) \pm \sqrt{\frac{p(x)R(k)^D}{Nh_1...h_D}} \\ &\approx \tilde{p}(x) - \frac{1}{2}\kappa_2(k) \sum_{d=1}^{D} h_d^2 \frac{\partial^2 \tilde{p}}{\partial x_d^2}(x) \pm \sqrt{\frac{\tilde{p}(x)R(k)^D}{Nh_1...h_D}} \end{aligned} \tag{2.13}$$

Con respecto al costo computacional, en principio el tiempo de evaluación es de orden N (ver Ec. 2.10). Sin embargo, es esperable que la contribución a $\tilde{p}(x)$ de los puntos $x^{(i)}$ lejanos a x sea despreciable, lo cual sugiere que sólo podrían evaluarse los más cercanos, según algún criterio. Si los datos están desordenados, la determinación de los vecinos más cercanos también es de orden N, pero existen estructuras para representar a X, como los *k-Dimensional Trees*, o *Ball Trees*, que reducen el orden a $log(N)$, aunque a costa de mayores requerimientos de memoria [10].

2.3.1. Muestreo

El muestreo con KDE se realiza con el siguiente algoritmo:

1. Se toma un $x^{(i)} \in X$ al azar.
2. Se obtiene la nueva muestra como $x = x^{(i)} + \Delta$, siendo Δ una perturbación aleatoria con la distribución del *kernel* y ancho de banda H, es decir:

$$P(\Delta) = |H|^{-1} K(H^{-1}\Delta) \tag{2.14}$$

De este modo la probabilidad de los datos generados reconstruye la expresión 2.10. Por otra parte, si las muestras en X están desordenadas, es posible tomarlas en orden, una tras otra, en lugar de al azar, volviendo al inicio de la lista si se llega al final. Esto trae notables mejoras computacionales.

2.3.2. Optimización del ancho de banda

Una de las principales dificultades del método KDE es determinar un ancho de banda adecuado. El objetivo más usual es minimizar el *mean integrated square error* (MISE), dado por la siguiente fórmula:

$$\begin{aligned} MISE(\tilde{p}, p) &= \int (\tilde{p}(x) - p(x))^2 dx \\ &\approx \int \left(Bias(\tilde{p}(x))^2 + Var(\tilde{p}(x)) \right) dx \\ &\approx \frac{\kappa_\nu(k)}{\nu!} \sum_{d=1}^{D} h_d^\nu R(\frac{\partial^\nu p}{\partial x_d^\nu}) + \frac{R(k)^D}{N h_1 ... h_D} \end{aligned} \tag{2.15}$$

Se puede ver, como es esperable, que mayores anchos de banda reducen la varianza pero aumentan el *bias*. Para el caso de $h_1 = ... = h_D = h$ puede despejarse el óptimo, obteniéndose:

$$\hat{h} = \left(\frac{(\nu!)^2 D R(k)^D}{2\nu \kappa_\nu^2(k) R(\nabla^\nu p)} \right)^{1/(2\nu+D)} N^{-1/(2\nu+D)} \tag{2.16}$$

Puede observarse que $\hat{h}$ depende de p, la cual es desconocida, por lo cual esta expresión no puede usarse directamente. De todos modos, la dependencia sólo es a través del parámetro integral $R(\nabla^\nu p) = \int (\sum_{d=1}^{D} \frac{\partial^\nu p}{\partial x^\nu})^2 dx$.

Cabe mencionar que tomar un ancho de banda h único sólo es adecuado si se aplica normalización de datos, lo cual usualmente se realiza escaleando cada variable con su desviación estándar. En ese caso el modelo para los datos no normalizados puede obtenerse con la expresión 2.6, y equivale a tomar un ancho de banda multidimensional, con cada h_d es proporcional a la desviación estándar, es decir:

$$h_d = h\sigma_d \tag{2.17}$$

Donde σ_d es la dispersión de $X_d = \{x_d^{(i)},\ x^{(i)} \in X\}$, y h el ancho de banda del modelo normalizado.

Existen 3 familias de técnicas de optimización de ancho de banda [11]:

- Reglas del dedo (*rules-of-thumb*)
- Métodos *plug-in*

- Validación cruzada

"Reglas del dedo"

Estas técnicas, también llamadas métodos de referencia normal, se basan en la expresión 2.16, y estiman el parámetro $R(\nabla^\nu p)$ suponiendo una distribución preestablecida para p (usualmente gaussiana). Los parámetros de dicha distribución se obtienen directamente de los datos. Un método muy utilizado de este tipo es la denominada *Silverman rule-of-thumb* [12]:

$$\hat{h_d} = C_{silv}(k, D)\sigma_d N^{-1/(2\nu+D)} \tag{2.18}$$

Donde C_{silv} es una constante dependiente del *kernel* y de la dimensionalidad del problema. Para un *kernel gaussiano* ($\nu = 2$) su expresión resulta:

$$C_{silv} = \frac{4}{2+D} \tag{2.19}$$

Métodos *plug-in*

Esta metodología consiste en emplear un h_0 inicial (semilla), típicamente con una regla del dedo, para estimar $R_0 = R(\nabla^\nu \tilde{p}_0)$, donde $\tilde{p}_0$ es la estimación de p mediante KDE con *bandwidth* h_0. De este modo de la fórmula 2.16 se obtiene h_1, con el cual se puede repetir el proceso, hasta converger. Existen algunas variantes de esta técnica que buscan mejorar las estimaciones de p por KDE, por ejemplo corrigiendo el *bias*.

En general, las técnicas en esta familia requieren el cálculo de derivadas de la distribución estimada, lo cual dificulta su implementación.

Validación cruzada

Estos métodos se basan en minimizar una distancia entre $\tilde{p}$ y p, como $MISE$ o KL, mediante una expresión aproximada que sólo dependa de los datos X y de $\tilde{p}$. En este sentido, una dificultad que tiene el método KDE, a diferencia de otras técnicas de estimación de densidad, es que la minimización de expresiones para dichas distancias usualmente converge a $H = 0$. Esto se debe a que, dado el conjunto X, la distribución más probable es en efecto un conjunto de deltas de Dirac en cada muestra, aunque esta no resulta de utilidad para el objetivo de la estimación de densidad y muestreo.

Para resolver este problema se emplea la validación cruzada, la cual consiste en dividir X en dos conjuntos X_{train} y X_{test}. Se utiliza X_{train} para construir el modelo KDE, y X_{test} como puntos de evaluación en la expresión utilizada para la distancia. Para la elección de X_{train} y X_{test} usualmente se utiliza un esquema *K-fold* (también conocido como *V-fold cross validation*) [13]. En el caso extremo de $K = N$ *folds*, dicho

método lleva a la *leave-one-out cross validation* (LOO), más confiable pero también más costosa computacionalmente.

La minimización del error de $\tilde{p}$ suele llevarse a cabo evaluando dicha distancia para cada H en una grilla. Es frecuente utilizar como semilla el ancho de banda obtenido por otro método (ej.: una regla del dedo), y construir la grilla de anchos de banda multiplicando dicha semilla por una grilla de factores, es decir:

$$H_j = f_j H_0, \;\; j = 1, ..., J \tag{2.20}$$

Donde H_j es cada ancho de banda evaluado, H_0 es la semilla, y f_j es cada factor (usualmente $f_j \approx 1$).

Una forma simple, y relativamente robusta, es utilizar la métrica CE, lo cual se denomina *Maximum Likelihood Cross-Validation* (MLCV) [14]. El método consiste en maximizar el siguiente factor de mérito FM:

$$FM = -CE(p, \tilde{p})_{MLCV}(H) \approx \frac{1}{N} \sum_{i=1}^{N} log(\tilde{p}_{-i}(x^{(i)})) \tag{2.21}$$

Donde $\tilde{p}_{-i}$ es la *leave-one-out estimation*, es decir la estimación construida con todo X excepto $x^{(i)}$. La gran ventaja de este método es que la función a optimizar puede evaluarse utilizando únicamente las funcionalidades básicas de KDE, es decir construir $\tilde{p}$ y evaluarla. Es posible utilizar un esquema *K-fold* en lugar de LOO.

Otro método muy usual de validación cruzada es la *Least Squares Cross-Validation*, la cual consiste en minimizar $MISE(p, \tilde{p})$ usando la siguiente aproximación:

$$MISE(p, \tilde{p})_{LSCV}(H) \approx \int p(x)^2 dx - 2\frac{1}{N} \sum_{i=1}^{N} \tilde{p}_{-i}(x^{(i)}) + \int \tilde{p}(x)^2 dx \tag{2.22}$$

El primer término no depende de H, por lo cual puede ignorarse. En el segundo término se emplea $\tilde{p}_{-i}$ en lugar de $\tilde{p}$ por razones mencionadas anteriormente. El último término no depende de p, por lo cual en principio es conocido. Para evaluarlo puede, por ejemplo, calcularse la integral por Monte Carlo (expresión 2.2).

Existen varios otros métodos que utilizan distintas aproximaciones para estimar $MISE(p, \tilde{p})$, buscando mejorar las propiedades de convergencia, como la tasa a la que se reduce la varianza de $\hat{H}$ cuando N crece, o la estabilidad del método.

KDE adaptativo

Si la densidad varía varios órdenes de magnitud entre diferentes regiones de interés, utilizar un único ancho de banda fijo podría no ser adecuado. Esto lleva a la metodología de ancho de banda variable, denominada KDE adaptativo.

Hay dos variantes de esta técnica [15]: ancho de banda dependiente del punto de evaluación x, y ancho de banda dependiente del dato $x^{(i)}$. El segundo método tiene las ventajas de que es más fácil asegurar la normalización, y que la determinación de los vecinos más cercanos (lo cual puede ser costoso) sólo debe realizarse una vez (aunque sobre todos los datos). Para este último método la expresión para el método KDE se convierte en:

$$\tilde{p}(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|H^{(i)}|}K\left(\left(H^{(i)}\right)^{-1}(x - x^{(i)})\right) \tag{2.23}$$

Donde $H^{(i)}$ es el ancho de banda (unidimensional o multidimensional) correspondiente a la muestra $x^{(i)}$.

Uno de los métodos de selección de ancho de banda más usuales en este caso es el de K Vecinos Cercanos (KNN), donde se toma un ancho de banda $H^{(i)}$ tal que la cantidad de datos $x^{(j)}$ con $|\left(H^{(i)}\right)^{-1}(x^{(i)} - x^{(j)})| < 1$ sea un número K fijo. Desde luego esto deja libertad dentro de las componentes de H, lo cual puede resolverse aplicando normalización de datos y tomando $H = h$ como un escalar. Cabe mencionar que este método depende fuertemente de K, el cual es un parámetro libre. Esto puede resolverse parcialmente si, en lugar de utilizar los anchos de banda obtenidos directamente, se los emplea como semilla en una optimización por validación cruzada.

2.3.3. Peso de muestras variable

Es posible considerar que cada muestra de entrenamiento tiene un peso estadístico, de modo que $X = \{(x^{(i)},\ w^{(i)}), x^{(i)} \in \mathbb{R}^D, w^{(i)} \in \mathbb{R}\}_{1 \leq i \leq N}$. En ese caso se definen los parámetros:

$$I = \sum_{i=1}^{N} w_i$$
$$p2 = \sum_{i=1}^{N} w_i^2$$

La expresión para la densidad estimada por KDE resulta:

$$\tilde{p}(x) = \frac{1}{I|H|}\sum_{i=1}^{N} w_i K(H^{-1}(x - x^{(i)})) \tag{2.24}$$

La expresión para el *bias* es la misma que en el caso anterior, pero para la varianza se debe reemplazar N por:

$$N_{eff} = \frac{I^2}{p2} \tag{2.25}$$

También se debe utilizar N_{eff} en lugar de N en las reglas del dedo.

2.4. Redes Neuronales

Hay diversos métodos donde se utilizan redes neuronales en la estimación de densidad y muestreo. En muchos casos están orientados a imágenes, donde la alta dimensión del problema dificulta métodos más simples. La mayoría de las técnicas (aunque no todas) se basan en emplear las redes para transformar los puntos $x \in \mathbb{R}^D$ a variables "latentes" $z \in \mathbb{R}^{D'}$, donde se buscará que la distribución sea una preestablecida, usualmente gaussiana estándar. Puede tenerse $D' < D$ (*dimensionality reduction*) o $D' = D$. Es decir que se busca obtener una f biyectiva tal que $f(z) = x$, y $p(z) = \mathcal{N}(0, 1)$, como se ilustra en la Figura 2.1. De este modo, para evaluar la densidad de probabilidad $p(x)$ se debe convertir x a z y evaluar la función conocida sobre ésta, mientras que para generar muestras se genera un z aleatorio y se lo convierte a x.

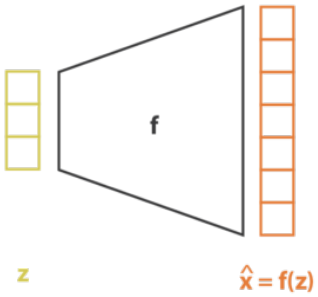


Figura 2.1: Transformación entre variables latentes y variables del problema, donde f se implementa a través de una red neuronal.

En general, todos los métodos en esta familia requieren un proceso de entrenamiento significativamente más complejo que los métodos anteriores. De hecho, usualmente el ajuste de cada modelo a un conjunto de muestras particular suele realizarse de forma manual, lo cual no es óptimo si se busca una herramienta de estimación de densidad y muestreo automática. De todos modos, a continuación se describirán las principales técnicas en esta familia, pues no se descarta su utilidad en trabajos futuros.

2.4.1. *Neural density estimators*

Si se enfoca el problema en evaluar la densidad, y no en obtener nuevas muestras, se puede utilizar la red simplemente para generar un valor de densidad de probabilidad $\tilde{p}(x)$ dado un $x \in \mathbb{R}^D$, y entrenarla para maximizar la log-probabilidad (minimizar CE) sobre X. Una de las dificultades que se presentan es que debe asegurarse por diseño que $\int \tilde{p}(x)dx = 1$. Los dos principales grupos de diseños de este tipo según [16] son los modelos autoregresivos (*autoregressive models*) y los flujos normalizadores (*normalizing flows*).

Autoregressive models

Estos métodos se basan en modelar la probabilidad $p(x)$ como probabilidades condicionales sucesivas, es decir:

$$\begin{aligned} p(x) &= p(x_1)p(x_2|x_1)p(x_3|x_1,x_2)...p(x_D|x_1,...,x_{D-1}) \\ p(x) &= \prod_{d=1}^{D} p(x_d|x_1,...,x_{d-1}) \end{aligned} \tag{2.26}$$

Se denominan autoregresivos porque la función $p(x_d|x_1,...,x_{d-1})$ se puede pensar como una regresión de x_d en función de $x_1,...,x_{d-1}$. Generar muestras nuevas es posible, aunque requiere D pasadas por la red.

Para lograr que cada probabilidad condicional esté normalizada, se pueden utilizar familias de funciones en las que todos sus miembros lo estén, con parámetros libres a ajustar. Los parámetros de cada función $\tilde{p}(x_d|x_1,...,x_{d-1})$ dependerán de $(x_1,...,x_{d-1})$, es decir:

$$\tilde{p}(x_d|x_1,...,x_{d-1}) = \tilde{p}_\Phi(x_d),\ \Phi = \Phi(x_1,...,x_{d-1}),\ \int \tilde{p}_\Phi(x_d)dx_d = 1\ \forall\Phi \tag{2.27}$$

Donde Φ representa los parámetros a ajustar de la función p_Φ

Una implementación notable de esta metodología es *Masked Autoencoder for Distribution Estimation* (MADE) [17].

Normalizing flows

En este caso se modela la variable x como $x = f(z)$, con $p(z) = \mathcal{N}(0,1)$ (*gaussiana* estándar). Entonces se tiene:

$$p(x) = p(f^{-1}(x)) \left| det\left(\frac{\partial f^{-1}}{\partial x}\right)\right| \tag{2.28}$$

La dificultad está en encontrar familias de funciones f fáciles de invertir y de calcular su Jacobiano. El objetivo para el entrenamiento suele ser la maximización de la log-probabilidad sobre los datos provistos.

Existen varias propuestas, entre las cuales se destacan los flujos autoregresivos, en los cuales se utiliza:

$$\begin{aligned} x_k &= z_k\sigma_k + \mu_k,\quad \sigma_k = \sigma_k(x_1,...,x_{k-1}),\ \mu_k = \mu_k(x_1,...,x_{k-1}) \\ \Rightarrow z_k &= (x_k - \mu_k)/\sigma_k \end{aligned}$$

Por su carácter autoregresivo, el Jacobiano de f^{-1} es triangular, por lo que su deter-

minante es la productoria de su diagonal:

$$\left| det\left(\frac{\partial f^{-1}}{\partial x}\right)\right| = \frac{1}{\prod_{k=1}^{D}\sigma_k} \tag{2.29}$$

El problema es que esto limita la distribución que pueden tener los x. Pero si los z se utilizan en el lugar de x en una nueva capa, y se repite el proceso para varias capas, se puede lograr modelar distribuciones arbitrariamente complejas para x.

Una implementación notable de esta técnica es *Masked Autoregressive Flow* (MAF) [18], en el cual se utilizó MADE como capa básica. Otra variante es *Inverse Autoregresive Flow* (IAF) [19], en la cual se toma $\sigma_k = \sigma_k(z_1, ..., z_{k-1})$, $\mu_k = \mu_k(z_1, ..., z_{k-1})$. Ambas implementaciones permiten la evaluación de $\tilde{p}(x)$ y la generación de muestras, pero, desde el punto de vista computacional, MAF está optimizado para la evaluación de $\tilde{p}(x)$, mientras que IAF lo está para la generación de muestras.

2.4.2. *Generative models*

Los modelos generativos, al contrario de los métodos de *Neural Density Estimation*, se enfocan en generar nuevas muestras, usualmente sin permitir evaluar la densidad estimada. Las dos principales familias de técnicas de este tipo, según [16], son los *Generative Adversarial Networks* (GAN) y los *Variational Autoencoders* (VAE).

Generative Adversarial Networks

Los GAN implementan una técnica proveniente de teoría de juegos, en la cual dos redes neuronales compiten entre sí. La primera de ellas, llamada generador, convierte un z sorteado de $p(z) = \mathcal{N}(0, 1)$ a x, buscando copiar la distribución p. La segunda, llamada discriminador, recibe alternadamente muestras creadas por el generador y muestras del conjunto de entrenamiento X, y para cada una busca determinar si son reales o generadas artificialmente. En la Figura 2.2 se ilustra este esquema. Si denominamos $f(z)$ a los datos generados en base a z, y $D(x)$ a la probabilidad (entre 0 y 1) estimada por el discriminador de que x provenga de X, el error del discriminación resulta:

$$E_d = \frac{1}{2}\mathbb{E}_{x\sim p(x)}(1 - D(x)) + \frac{1}{2}\mathbb{E}_{z\sim p(z)}(f(z)) \tag{2.30}$$

El entrenamiento consiste entonces en buscar que el generador maximice el error de discriminación ("engañando" al discriminador), y que el discriminador lo minimice. Desde luego esto debe hacerse de forma equilibrada, balanceando el entrenamiento de ambas redes.

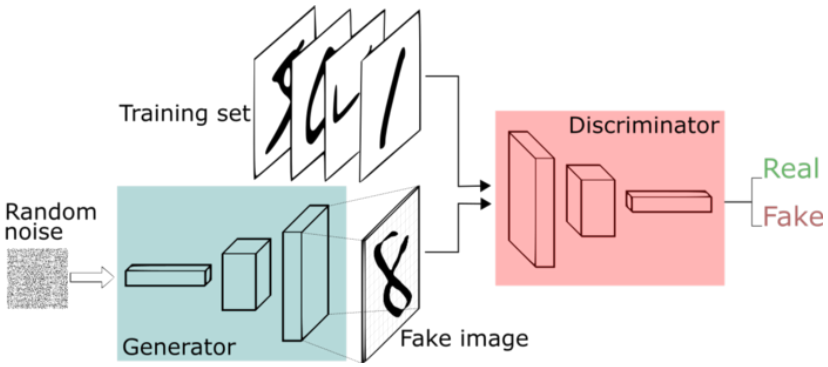


Figura 2.2: Esquema de un GAN.

Variational Autoencoders

Los VAE son una variante de los *autoencoders* tradicionales, adaptados para la generación de datos. En un *autoencoder*, la variable $x \in \mathbb{R}^D$ es codificada a $z \in \mathbb{R}^{D'}$, tipicamente con $D' < D$, implementando tanto el codificador f como el decodificador g con redes neuronales, entrenadas para cumplir $g = f^{-1}$ lo más posible. En este caso, al no haber control sobre la distribución que tendrá z, no puede asegurarse que un valor de z muestreado al azar tenga sentido al convertirlo a x. Para resolver este problema se agrega un carácter aleatorio a la arquitectura: se reemplaza $g(x)$ por $\tilde{p}(z|x)$ y $f(z)$ por $\tilde{p}(x|z)$, como se esquematiza en la Figura 2.3. El objetivo de entrenamiento de la red neuronal será entonces doble:

- Reconstrucción: Maximizar $\mathbb{E}_{z\sim\tilde{p}(z|x)}\tilde{p}(x|z)$.
  Esto significa que la distribución $\tilde{p}(x|z)$, para un z muestreado de $\tilde{p}(z|x)$ debe tener valores altos (pico) en x.

- Regularización: Minimizar $KL(\tilde{p}(z|x), p(z))$.
  Esto significa que se busca acercar la distribución $\tilde{p}(z|x)$ a la distribución $p(z) = \mathcal{N}(0, 1)$ impuesta en el espacio latente.

Las expresiones de los objetivos provienen de la metodología *Variational Inference*, de la cual se obtiene que el objetivo final a maximizar es el denominado *evidence lower bound*:

$$\begin{aligned}\mathcal{L}(x) &= \mathbb{E}_{z\sim\tilde{p}(z|x)}(log(\tilde{p}(x|z))) - KL(\tilde{p}(z|x), p(z)) \\ &= \mathbb{E}_{z\sim\tilde{p}(z|x)}\left(\frac{\tilde{p}(x|z)p(z)}{\tilde{p}(z|x)}\right) \\ &= \mathbb{E}_{z\sim\tilde{p}(z|x)}\left(-\frac{||x-\mu_x(z)||^2}{2\sigma_x}\frac{p(z)}{\tilde{p}(z|x)}\right)\end{aligned} \tag{2.31}$$

Donde usualmente $p(z) = \mathcal{N}(0, 1)$. En la última expresión se usó $\tilde{p}(x|z) = \mathcal{N}(\mu_x(z), \sigma_x)$, con σ_x un parámetro fijo. La función $\mathcal{L}$ resulta entonces un balance entre la reconstruc-

ción y la regularización, regulado por el parámetro σ_x, donde la optimización de uno necesariamente lleva al deterioro del otro. Las esperanzas pueden evaluarse por Monte Carlo, usualmente con un único muestreo. Por último, el objetivo se evalúa como el promedio de $\mathcal{L}(x^{(i)})$, $x^{(i)} \in X$.

Los VAE tienen propiedades interesantes relacionadas con la regularización de la distribución en el espacio latente. Si el entrenamiento converge adecuadamente, la métrica en el espacio latente (direcciones principales, distancia entre valores de z) puede tener significados relevantes con respecto a la distribución en x. Esto se representa esquemáticamente en la Figura 2.4. Cabe mencionar que estas propiedades no son de particular utilidad a los objetivos de este trabajo.

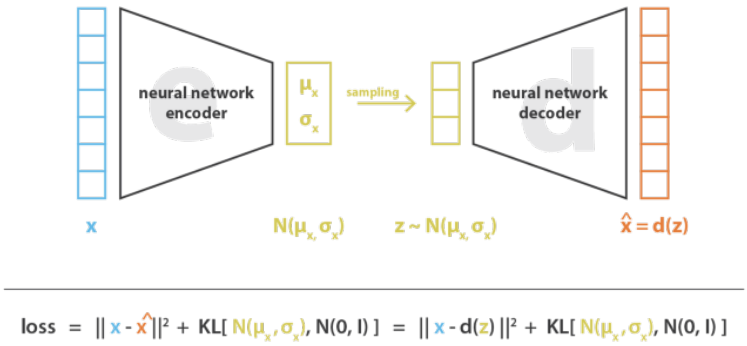


Figura 2.3: Esquema de un VAE.

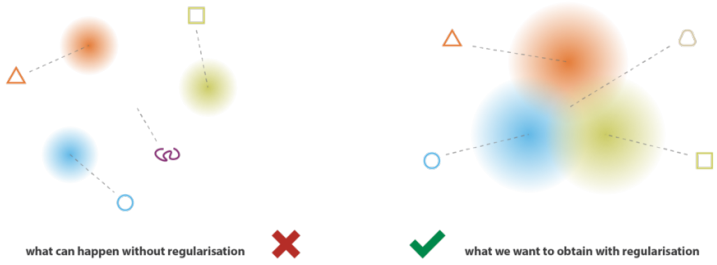


Figura 2.4: Comparación entre un caso con baja regularización, pero potencialmente buena reconstrucción (izq.) y uno regularizado, a costa de una menor reconstrucción.

2.5. Estudios de factibilidad

Se comparó la factibilidad del método KDE y de redes neuronales mediante implementaciones sencillas. Dentro de los métodos de redes neuronales, se eligió estudiar las GANs. Se implementaron ambos modelos en lenguaje Python, utilizando la biblioteca `SciKit-Learn` para el modelo KDE, y `tensorflow-keras` para el GAN. Como conjunto de muestras se utilizó una lista de 10^5 neutrones, extraída de la fuente de *tracks* caracterizada en la Sección 4.4.

Con respecto al método KDE, se logró representar la distribución de la lista de partículas de acuerdo a lo esperado. Se implementó exitosamente la optimización de ancho de banda por validación cruzada (MLCV), y fue posible la generación de nuevas muestras respetando la distribución estimada. Se observó que, si bien dicho método de selección de ancho de banda puede resultar relativamente costoso computacionalmente, asegura la obtención de resultados apropiados luego del tiempo de ejecución.

El entrenamiento del modelo GAN, por el contrario, presentó varias dificultades. Para empezar, los modelos basados en redes neuronales, debido al alto costo computacional requerido, necesitan el empleo de cómputo paralelizado en GPU. Por otra parte, su entrenamiento requiere casi inevitablemente la participación activa del usuario, para la optimización de hiperparámetros. Esto usualmente se realiza, al menos parcialmente, de forma manual, debido que no existen métodos automatizados que funcionen de manera general. En particular, en este caso, el mejor modelo que se logró entrenar presentó una performance notablemente inferior a la del KDE, con costos de entrenamiento notablemente superiores. Si bien el modelo GAN es particularmente complejo dentro de las redes neuronales, en general todos los métodos en esta familia presentan para el entrenamiento una combinación de alto costo computacional y alto costo en horas de ingeniería.

2.6. Comparación y elección

En el presente capítulo se presentaron y describieron cuatro familias de métodos de estimación de densidad y muestreo. Se buscó presentarlos en orden de complejidad creciente, es decir que el método considerado más simple resulta ser el de histogramas, mientras que el más complejo viene a ser el de redes neuronales. En general, una mayor complejidad está asociada con una mejor calidad del modelo de estimación de densidad y muestreo, pero con una mayor dificultad para optimizarlo adecuadamente. Está tendencia posee, a saber, dos excepciones:

- En el método de histogramas, que debería ser el más sencillo de optimizar, la elección del conjunto de discretizaciones es en realidad un proceso relativamente complejo. Desde luego esto puede resolverse aplicando restricciones, por ejemplo tomando un ancho de *bins* constante, pero esto penaliza la calidad de los modelos obtenidos.

- En caso de que el ajuste del modelo no sea óptimo, lo cual podría ser frecuente, los modelos de redes neuronales, así como los modelos paramétricos multimodales, pueden resultar es distribuciones completamente alejadas de la distribución a modelar, debido a fenómenos de inestabilidades computacionales. En comparación, los histogramas y KDE son más robustos con respecto a ajustes no óptimos.

Teniendo en cuenta las ventajas y desventajas de los distintos métodos, descritas en las secciones correspondientes, las consideraciones apenas mencionadas, y los resultados del estudio de factibilidad, se determina que el método más adecuado para el presente trabajo es el *Kernel Density Estimation*. El mismo es una generalización natural tanto de los histogramas como de los modelos paramétricos, con mejoras considerables en la calidad de las estimaciones, y sin las dificultades notablemente mayores de las redes neuronales.

Más allá de las numerosas ventajas del método KDE, resulta importante conocer las principales dificultades que se deberá afrontar al implementar dicha técnica. A continuación se describen los dos puntos considerados más relevantes, junto con posibles estrategias para reducir su impacto.

- Costo computacional para la evaluación relativamente alto: Como ya fue mencionado, la evaluación de la densidad estimada es una operación de orden N, siendo N la cantidad de partículas en la lista, pero puede reducirse a $log(N)$ si se utilizan estructuras como los *KDTree* o *BallTree* [10] para representar la lista. De todos modos cabe recordar que el objetivo principal de las fuentes distribucionales es la generación de muestras, y no la evaluación de densidad.

- Costo en memoria relativamente alto para el almacenamiento del modelo: A diferencia de otras técnicas de estimación de densidad, KDE requiere el almacenamiento de la lista de *tracks* como parte del modelo. La utilización de formatos binarios, evitando niveles de precisión numérica innecesariamente altos, ayuda a mitigar esta dificultad.

Si en trabajos futuros se deseara implementar una fuente distribucional mediante redes neuronales, se recomienda implementar un flujo normalizador (ver 2.4.1), tomando como referencia el modelo IAF [19]. Esto se debe a la relativa simplicidad de esta arquitectura, y a su capacidad de tanto evaluar densidad como generar muestras, en contraste con modelos generativos como el GAN, que arrojaron numerosas dificultades en el estudio de factibilidad.

Capítulo 3

Desarrollo de herramientas computacionales

Tomando como punto de partida la elección del método KDE, se desarrolló una herramienta que utilice dicha técnica para implementar fuentes de radiación. Para la misma se plantean los siguientes requerimientos:

- Debe ser capaz de ajustar un modelo KDE a una lista de partículas y optimizar ancho de banda (BW), lo más automáticamente posible.
- El modelo optimizado debe ser capaz de producir nuevas partículas que respeten la distribución estimada. Para ello existen dos posibilidades:
  - Fuente externa *on-the-fly*: La fuente genera las partículas de fuente a medida que el código Monte Carlo se las demanda. El formato de la fuente y de las partículas generadas debe adecuarse a cada código en particular.
  - Nueva lista de partículas: Se utiliza la herramienta para generar una nueva lista de partículas con una longitud fijada por el usuario, usualmente mayor que la longitud original. Luego se utiliza la lista generada como input en una simulación Monte Carlo.

Por otra parte, existen otras funcionalidades, que, si bien no son requisitos centrales en la herramienta a desarrollar, resultan deseables, pues facilitan el trabajo con fuentes distribucionales. A continuación se describen algunas de ellas:

- Gráficos de las distribuciones: Si bien no es esencialmente necesario para producir partículas, resulta muy útil poder graficar las distribuciones estimadas, incluidas las correlaciones.
- Análisis estadístico, con el objetivo de analizar el grado de convergencia de una lista de *tracks*, y determinar si su cantidad es apropiada.

- Control de peso de partículas, *source biasing*: Las partículas en una fuente poseen diferentes pesos estadísticos. Sin embargo, es deseable que al muestrear partículas de fuente estas tengan peso unitario. O más aún, controlando con cierto criterio su peso se puede favorecer la estadística de ciertas regiones del espacio de fases de la fuente.

- Controles de autoconsistencia: Para verificar el correcto ajuste de un modelo KDE, resulta útil poseer un módulo que compare magnitudes integrales entre la fuente KDE y la lista de *tracks* original, orientado a casos de aplicación usuales.

- Registro de magnitudes integrales: Las intensidades totales de las fuentes permiten llevar un registro de las corrientes involucradas en una simulación, por lo que es deseable poder calcularlas de forma sencilla.

La herramienta desarrollada se denominó KDSource, y cumple con todos los requerimientos y funcionalidades deseables descritas anteriormente. Se basa en los lenguajes Python, C y *bash script*, y permite generar fuentes para varios códigos Monte Carlo, con una particular orientación hacia McStas y TRIPOLI-4. En concordancia con estos dos códigos, la herramienta está orientada principalmente a dos tipos de aplicaciones: haces de neutrones y cálculos de blindajes. En este ultimo caso es posible el cálculo acoplado de neutrones y fotones, así como el modelado de fuentes de activación.

En las siguientes secciones se describen las diferentes herramientas externas utilizadas, módulos incluidos, y conceptos en los cuales se basa el modelado de fuentes KDE con la herramienta KDSource. Para una documentación más detallada de las diferentes funcionalidades, ver el Apéndice A. Dicha documentación también se encuentra presente en el manual de KDSource, incluido en la distribución del paquete.

3.1. Herramientas externas

3.1.1. MCPL: *Monte Carlo Particle Lists*

MCPL [20] es un formato general de listas de partículas, el cual se eligió como herramienta auxiliar para el tratamiento de listas de *tracks*. Es un formato binario, en el cual cada partícula en la lista es caracterizada por su tipo, energía, posición, dirección, tiempo, y opcionalmente polarización. Se permite una gran cantidad de tipos de partículas diferentes gracias a la utilización del código PDG [21], el cual codifica, mediante un número entero, desde partículas elementales hasta iones pesados, incluyendo desde luego neutrones y fotones. Cabe mencionar que el formato binario reduce los requerimientos de memoria, lo cual, como se explicó en la Sección 2.6, es un aspecto importante en el método KDE.

Las listas de partículas se acceden y manipulan a través de APIs en Python y C, así como mediante la aplicación de línea de comando `mcpltool`. Toda la herramienta es de código abierto.

MCPL se comunica con distintos códigos Monte Carlo a través de los denominados *hooks*. En la mayoría de los casos, estos *hooks* son aplicaciones de línea de comando que permiten la conversión en ambos sentidos entre el formato MCPL y un formato de lista de partículas específico del código en cuestión. A continuación se listan los códigos con *hooks* implementados, los cuales se esquematizan el la Figura 3.1:

- MCNP (versiones MCNPX, MCNP5 y MCNP6) [22–24]
- PHITS [25]
- GEANT4 [26–28]
- McStas [4–6]
- McXtrace [29]

Además, aprovechando el carácter *open-source* de MCPL, se agregó la conversión del formato de TRIPOLI-4 [7] a MCPL. La conversión en el otro sentido no es requerido gracias al muestreo *on-the-fly* el cual se describe en la Subsección 3.4.1. También se añadió la conversión a MCPL desde el formato de texto SSV, lo cual facilita la creación de listas MCPL desde Python. Por último, si bien está implementada la conversión desde el formato binario de listas de partículas de MCNP (SSW/SSR), se añadió la conversión desde el formato PTRAC, el formato análogo de tipo ASCII.

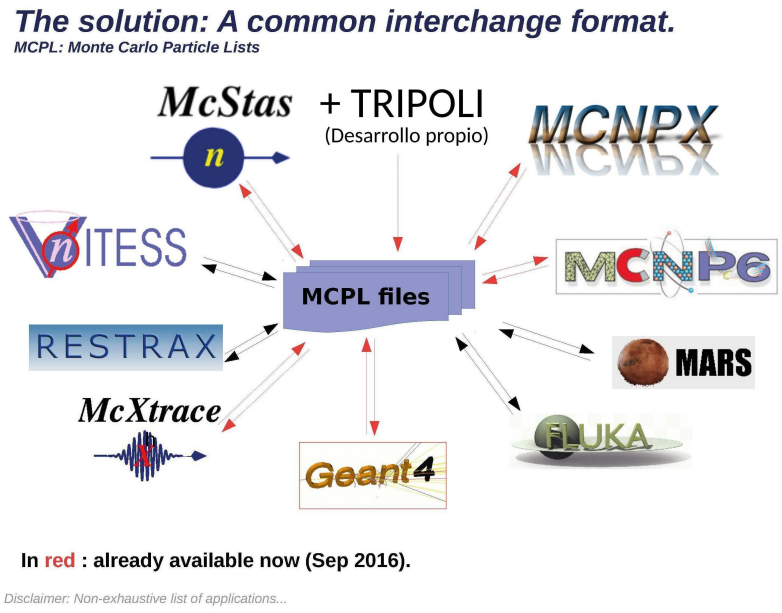


Figura 3.1: Esquema de la interacción entre MCPL y los códigos Monte Carlo compatibles. Las flechas rojas representan los *hooks* implementados, mientras que las negras indican los planificados. TRIPOLI fue añadido mediante desarrollo propio.

3.1.2. Librerías de KDE en Python

Existe una gran cantidad de bibliotecas en Python que implementan el método KDE, por lo cual se decidió escoger una de ellas para dicha tarea. Se analizaron las opciones listadas a continuación, consideradas las implementaciones más consolidadas. Se presenta en cada ítem la biblioteca y el modelo (clase).

- SciPy [30]: `gaussian_kde`
- Scikit Learn [31]: `KernelDensity`
- Statsmodels [32]: `KDEMultivariate`
- KDEpy [33]: `TreeKDE` y `FFTKDE`

En la Tabla 3.1 se resumen las principales características de cada biblioteca, y la disponibilidad de las funcionalidades más importantes. Las mismas se describen a continuación:

- Múltiples *kernels*: Disponibilidad de múltiples funciones *kernel* (ej.: *gaussiano*, *epanechnikov*, uniforme, triangular, etc. [8]). Si bien en este trabajo solo se utilizó el *kernel gaussiano*, es deseable contar con la posibilidad de emplear otros en el futuro.
- Ancho de banda multidimensional: Soporte para diferentes anchos de banda h_d para cada dimensión d (caso H diagonal en Ec. 2.10). Su no disponibilidad equivale a tomar todos los h_d como iguales.
- Muestras pesadas: Soporte para factores de peso w_i para cada partícula i (ver Ec. 2.24). Su no disponibilidad equivale a tomar $w_i = 1$ para todo i.
- Ancho de banda automático: Implementación de métodos automáticos de selección de ancho de banda. Las familias de técnicas más usuales son los métodos de referencia normal (NR), o "reglas del dedo", como la Regla de Silverman [12], y los métodos de validación cruzada (CV), usualmente la MLCV [14], con esquemas KFold o *leave-one-out* (LOO) [13]. Ver Subsección 2.3.2 para más detalles.
- Ancho de banda variable: Soporte para diferentes anchos de banda $h^{(i)}$ para cada partícula i en la lista (ver Ec. 2.23). Su no disponibilidad equivale a tomar todos los $h^{(i)}$ como iguales.
- Evaluación libre: Posibilidad de evaluar la densidad estimada en cualquier punto, en contraste con implementaciones en las cuales solo es posible evaluar el método simultáneamente en una grilla regular de puntos.

- Aceleración: Método para reducir el tiempo de evaluación del KDE, teniendo en cuenta su relativamente alto costo computacional. Los métodos más comunes son *KDTrees* o *BallTrees* [10], o técnicas basadas en la *Fast Fourier Transform* (FFT) [34].

Un factor adicional que se consideró es cuán fácilmente se puede resolver la falta de alguna de dichas funcionalidades. En particular, el ancho de banda unidimensional puede resultar equivalente a uno multidimensional a través de la normalización de datos. Los métodos de optimización de ancho de banda también pueden ser implementados externamente. Por otra parte, un modelo KDE no adaptativo no puede convertirse en adaptativo mediante agregados externos.

Dadas las ventajas y desventajas listadas, y teniendo en cuenta las últimas consideraciones, resulta claro que la implementación más adecuada es `TreeKDE`, de la biblioteca `KDEpy`. La principal razón es que es la única que incluye KDE adaptativo, además de una aceleración razonable de los tiempos de evaluación a través de la estructura *KDTree*.

	SciPy	Scikit-Learn	Statsmodel	KDEpy. TreeKDE	KDEpy. FFTKDE
Múltiples *kernels*	No	Sí	Sí	Sí	Sí
BW multidimensional	Sí	Sí	Sí	Sí	Sí
Muestras pesadas	No	Sí	Sí	Sí	Sí
BW automático	NR	CV	NR, CV (LOO)	NR	NR
BW variable	No	No	No	Sí	No
Evaluación libre	Sí	Sí	Sí	Sí	No
Aceleración	No	KDTree, BallTree	No	KDTree	FFT

Tabla 3.1: Comparación entre las principales liberías de KDE en Python. Las celdas verdes significan que la funcionalidad correspondiente está adecuadamente implementada, las amarillas significan que está implementada pero no de forma óptima, y las rojas que está pobremente implementada o no incluida.

3.2. Contenidos del paquete KDSource

La herramienta KDSource cuenta con los siguiente componentes:

- Biblioteca en Python, para estimación de densidad. Aquí se encuentran las herramientas necesarias para crear y optimizar una fuente KDSource en base a una lista de partículas. Se puede además analizar su estadística, generar gráficos de las

distribuciones, y exportar la fuente creada, para su uso en los otros componentes de KDSource.

- Biblioteca en C, para muestreo. Aquí se encuentran las herramientas necesarias para muestrear nuevas partículas, en base a una fuente KDSource previamente creada.

- Aplicación de línea de comando: `kdtool`. Permite ejecutar un re-muestreo de partículas, en base a una fuente KDSource previamente creada, obteniéndose una lista de partículas de longitud arbitraria. También permite acceder de forma simple a otras utilidades incuidas en el paquete.

- Plantillas y archivos útiles. Facilitan las operaciones más usuales con el paquete KDSource. Se incluyen archivos Jupyter Notebook con las operaciones más usuales de la biblioteca de Python, así como *scripts* y otros componentes para ejecutar algunos códigos Monte Carlo en acople con KDSource.

En la Sección 3.3 se describen los principales objetos definidos en las bibliotecas, mediante los cuales se modelan y manipulan las fuentes KDE. En los Apéndices A.2, A.3 y A.4 se presenta la documentación detallada de la aplicación de línea de comando y de las bibliotecas. Por otra parte, en la Sección 3.4 se describe el flujo de trabajo requerido para emplear la herramienta KDSource en una simulación Monte Carlo.

3.3. El formato de fuente KDSource

La herramienta KDSource cuenta con bibliotecas para modelar fuentes distribucionales de partículas en Python y C. En ambos casos se define la estructura `KDSource`, la cual a su vez se compone de dos subestructuras: `PList` y `Geometry`. Éstas modelan los dos componentes fundamentales de una fuente KDSource: La lista de partículas y su geometría. En Python, la técnica KDE se implementa a través de la biblioteca `KDEpy` [33].

La estructura `KDSource` posee además el ancho de banda del modelo KDE, el cual es de la siguiente forma:

$$h_d^{(i)} = s_d \bar{h}^{(i)} \tag{3.1}$$

Donde $h_d^{(i)}$ es el ancho de banda correspondiente a la i-ésima partícula en la lista, y la dimensión d. $\bar{h}^{(i)}$ es el denominado ancho de banda normalizado, unidimensional pero variable para cada partícula, mientras que s_d es un factor de escaleo (*scaling*) para la dimensión d (normalización de datos). El ancho de banda que emplea la biblioteca `KDEpy` se corresponde con la componente unidimensional $\bar{h}^{(i)}$, mientras que los escaleos s_d se aplican externamente.

3.3.1. Listas de partículas

Las listas de partículas utilizadas en KDSource utilizan el formato MCPL [20]. El mismo permite la comunicación con múltiples códigos Monte Carlo, listados en la Subsección 3.1.1. Esto quiere decir que es posible convertir las listas de partículas registradas por cualquiera de estos códigos a MCPL, y viceversa. El *software* asociado al formato MCPL está incluido en la distribución de KDSource, incluyendo funcionalidades extra con respecto a la distribución original, como la posibilidad de comunicación con TRIPOLI-4.

La estructura `PList` empleada en las bibliotecas administra la comunicación con el archivo MCPL. Además de la lectura y escritura, se incluye la posibilidad de aplicar una traslación y rotación a las partículas inmediatamente despues de leerlas, lo cual puede ser útil al acoplar simulaciones con distinto sistema de referencia.

3.3.2. Geometría

En el formato de MCPL, el vector de fase $\mathbf{p}$ que define a una partícula es el siguiente (ignorando el tiempo y polarización):

$$\mathbf{p} = (E, x, y, z, u_x, u_y, u_z) \tag{3.2}$$

Donde E es la energía, (x, y, z) son las coordenadas de la posición, y (u_x, u_y, u_z) es el vector unitario de dirección (en un sistema de referencia dado). Dependiendo de la geometría de fuente y otras características específicas del problema, este vector de fase podría no ser el más apropiado para la aplicación del método KDE. Por ello se introduce el concepto de parametrización de variables, el cual es llevado a cabo por el objeto `Geometry` (Geometría), y se describe a continuación.

A pesar de su nombre, la estructura `Geometry` no sólo administra la geometría de fuente, sino también la manera en que se debe tratar la energía y dirección de las partículas. El objetivo fundamental de esta estructura es convertir el conjunto de parámetros que definen una partícula, es decir energía, posición y dirección, a un vector de parametrización (variables parametrizadas), más adecuado para la aplicación del método KDE (ej.: (E, x, y, μ, ϕ)). Dicho vector es el que se utilizará como x en las expresiones de la Subsección 2.3.

Para dar versatilidad al tratamiento energético, espacial y angular deseado, la estructura `Geometry` posee a su vez un conjunto de subestructuras de tipo `Metric`. Usualmente se cuenta con tres de éstas: una para la energía, una para la posición, y una para la dirección. Cada `Metric` define la métrica con la que se tratará cada conjunto de variables. Los objetos `Metric` pueden elegirse del conjunto de métricas implementadas:

- `Energy`: tratamiento simple de la energía, sin transformaciones.

- `Lethargy`: emplear letargía en lugar de energía.
- `Vol`: tratamiento espacial volumétrico.
- `SurfXY`: tratamiento espacial plano en XY.
- `Guide`: geometría de guía de sección rectangular, con tratamiento de los ángulos basado en la superficie de cada espejo. Las variables de parametrización se muestran en la Figura 3.2.
- `Isotrop`: tratamiento simple de la dirección, basado en el versor unitario de dirección y un único ancho de banda.
- `Polar`: tratamiento de la dirección en base a los ángulo θ (distancia angular a la dirección $\hat{z}$) y ϕ (acimut medido desde la dirección $\hat{x}$).
- `PolarMu`: igual a `Polar`, pero con $\mu = cos(\theta)$ en lugar de θ.

Cabe mencionar que la aplicación del *kernel gaussiano* sobre la dirección con métrica `Isotrop` (vector tridimensional unitario) resulta en la denominada distribución de von Mises-Fischer [35].

En la biblioteca de Python, la función principal tanto de `Geometry` como de `Metric` es la de transformar las partículas en el formato de partícula de MCPL al vector de parametrización, en formato de `numpy.array`. Esto se logra a través de las funciones `transform` e `inverse_transform`.

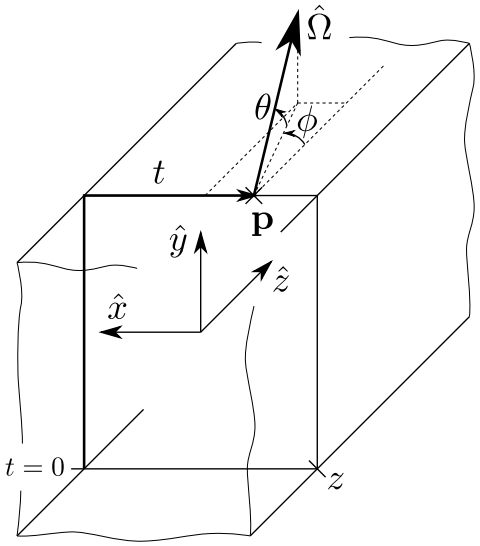


Figura 3.2: Esquema de las variables de parametrización de la geometría `Guide`: (z, t, μ, ϕ). Para una partícula $\mathbf{p}$ escapando de una guía, la variable z es la distancia desde el inicio de la guía hasta la partícula, t representa su ubicación medida circulando alrededor de la guía, para z constante, mientras que $\mu = cos(\theta)$ y ϕ son los parámetros polares de la dirección $\hat{\Omega}$.

En la biblioteca de C, por su parte, considerando que ésta se enfoca en el muestreo de partículas, lo cual se logra mediante la perturbación (ver 2.3), la función principal de `Geometry` y `Metric` es la de perturbar partículas, respetando la métrica correspondiente y con los anchos de banda provistos. Esto se logra con las funciones `[MetricName]_perturb`, donde `[MetricName]` se debe reemplazar por el nombre de cada métrica.

En algunos casos, las variables de ciertas métricas poseen fronteras, es decir valores que no deben superar (o caer por debajo), sea por motivos matemáticos (ej.: variables μ ó θ) o por decisión del usuario (ej.: fuentes limitadas espacialmente). Esto representa una dificultad para el método KDE, ya que el *kernel gaussiano* no está limitado espacialmente. La técnica elegida en estos casos es la siguiente: si al perturbar una partícula esta cae por fuera de una frontera, es reflejada con respecto a dicha frontera. De este modo no solo se logra que las partículas siempre caigan dentro de la región permitida, sino que además se mantiene el valor de la distribución cerca del borde. Esto quiere decir que, para distribuciones uniformes, la técnica de reflexión permite que la distribución estimada no presente ninguna perturbación en el borde. Distribuciones que presentan un gradiente no nulo en la frontera sí presentarán un efecto de borde, tendiente a anular dicho gradiente.

El objeto `Geometry` también incluye (opcionalmente) una traslación y rotación espacial. Esto permite modelar fuentes ubicadas en dististas posiciones y con distintas orientaciones con respecto al sistema de coordenadas. Por ejemplo, permite modelar una fuente en el plano YZ mediante la métrica `SurfXY`, a través de una rotación de 90 grados en el eje Y.

3.3.3. Archivos de parámetros XML

Las fuentes creadas en Python, luego de la optimización, pueden guardarse en un archivo de parámetros en formato XML. En el mismo se registran los parámetros que definen las estructuras `PList` y `Geometry` que componen la fuente `KDSource`, así como el *path* al archivo MCPL a utilizar. Este archivo de parámetros puede luego utilizarse para reconstruir la fuente mediante la biblioteca en C, o bien para re-muestrear directamente mediante la aplicación de línea de comando.

Si el ancho de banda del modelo es constante, su valor también se guarda en el mismo archivo XML. Si, por el contrario, el ancho de banda es variable (KDE adaptativo), es decir que se representa con un *array* de un valor por partícula, el mismo se guarda en un archivo separado, en formato binario. Para maximizar el ahorro de memoria, especialmente importante en listas de partículas largas, se utiliza el formato de punto flotante de simple presición (32 bits), sin separación entre valores.

3.3.4. Optimización de ancho de banda

En la biblioteca de Python se complementó la librería `KDEpy` con la técnica de normalización de datos y un módulo con los siguientes métodos de optimización de ancho de banda:

- Regla de Silverman, considerando la multidimensionalidad y peso de muestras variable [12].
- K Vecinos más Cercanos (KNN) [15].
- Validación Cruzada de Máxima Probabilidad (MLCV) [14].

En estos métodos se aprovecharon algunas funcionalidades de `Scikit-Learn` (búsqueda de vecinos, *K-folding*), y paralelización en CPU mediante la biblioteca `joblib`.

El algoritmo de optimización de ancho de banda recomendado es el acople de métodos KNN+MLCV, lo cual consiste en la optimización por MLCV de una semilla obtenida por KNN. Dado el relativamente alto costo computacional del método MLCV, es conveniente aplicarlo sobre un subconjunto de N_{CV} datos, y luego extenderlo a la lista completa de N partículas mediante la Regla de Silverman (ver Subsección 2.3.2). La expresión resultante para el ancho de banda es la siguiente:

$$h_d^{(i)} = C s_d d_K^{(i)} \left(\frac{N}{N_{CV}} \right)^{-\frac{1}{4+D}} \tag{3.3}$$

Donde $h_d^{(i)}$ es el ancho de banda correspondiente a la i-ésima partícula en la lista, y la dimensión d, $d_K^{(i)}$ representa la distancia desde la i-ésima muestra al K-ésimo vecino, s_d es un factor de escaleo (*scaling*) para la dimensión d (normalización de datos), y C es el factor de escaleo general optimizado por MLCV con N_{CV} datos. La semilla del método MLCV deben ser las N_{CV} distancias $d_K^{(i)}$ correspondientes a los N_{CV} datos empleados para la optimización. El parámetro K es elegido por el usuario, aunque tiene poca influencia en el resultado final gracias al método MLCV, que actúa compensando las variaciones en $d_K^{(i)}$ causadas por diferentes elecciones del valor de K.

Por *default*, los factores de escaleo s_d se calculan como la desviación estándar σ_d de la lista de partículas sobre cada dimensión, de acuerdo a la técnica de normalización de datos. Sin embargo, fijando manualmente factores de escaleo distintos de σ_d, para una variable dada, es posible controlar la calidad de la estimación sobre esa dimensión. En particular, si se toma $s_d = \sigma_d/\alpha_d$ en lugar de σ_d, el factor α_d puede interpretarse como un factor de importancia, o grado de prioridad, de la variable d. Un α_d mayor a 1 lleva a una mejor estimación de la distribución en la dimensión d, a expensas de una disminución en la calidad sobre las otras variables.

Por otra parte, en algunas situaciones hay razones físicas para fijar manualmente el escaleo de una dimensión. Por ejemplo, para espectros de emisión γ discretos, puede emplearse un *scaling* de energía $s_E = 0$ para evitar un suavizado. O para distribuciones isotrópicas es posible fijar $s_d = \infty$ para todas las componentes de la dirección. En estos casos, la modificación manual de los factores de escaleo debe llevarse a cabo después de la optimización del ancho de banda, por los problemas numéricos que puede generar el uso de valores nulos o infinitos en dicho proceso. De todos modos, se recomienda antes de la optimización emplear factores de importancia muy bajos, 10^{-3} por ejemplo, para que el algoritmo priorice el ajuste de las demás variables. Esto, por otra parte, facilita en gran medida el ajuste, pues supone en la práctica una reducción en la dimensionalidad del problema.

3.4. Flujo de trabajo

El esquema del flujo de trabajo típico con la herramienta KDSource se esquematiza en la Figura 3.3. Se parte de una lista de partículas inicial, la cual puede, o no, haber sido creada en una simulación anterior, y luego se ejecuta cierto número de simulaciones Monte Carlo. Cada una de ellas emplea una fuente de partículas KDSource basada en la lista de partículas registrada inmediatamente antes, y registra una nueva lista de partículas para la etapa siguiente. Dependiendo del problema, pueden obtenerse resultados útiles en todas las simulaciones (por ejemplo, al construir un mapa de dosis), o bien sólo en la última (por ejemplo, al calcular el flujo a la salida de un haz). Cada simulación cubre una región de la geometría modelada progresivamente más lejana a la fuente inicial, alcanzando distancias que no serían alcanzables en una sola corrida.

Cada construcción de una fuente KDSource en base a una lista de partículas se realiza mediante la API en Python, y culmina al exportar un archivo de parámetros XML. Se provee una plantilla con el código requerido para dicha tarea en el archivo `preproc_tracks.ipynb`.

Es recomendable ejecutar dos veces cada etapa de simulación, una de ellas utilizando la fuente KDE de la manera usual, y en la otra muestreando partículas directamente de la lista de *tracks*. Esto puede lograrse de forma simple fijando el argumento `use_kde=0` en la función de muestreo, empleando la misma fuente `KDSource`. El objetivo de dicha repetición del cálculo es verificar la compatibilidad de los resultados obtenidos con ambas fuentes.

De acuerdo a lo explicado en el Apéndice B, las fuentes de *tracks* pueden añadir mayor "ruido" a la simulación, pero no introducen ningún sesgo (*bias*). Las fuentes KDE, gracias a su mejor estadística, permiten obtener resultados más detallados en zonas lejanas a la fuente, y más suaves, pero corren el riesgo de introducir errores sistemáticos debidos al sesgo de suavizado. Si los resultados que es posible obtener

con ambas simulaciones (cercanos a la fuente) coinciden con ambas fuentes, puede considerarse que el sesgo de la fuente KDE es suficientemente bajo, y "confiar" en los resultados más lejanos.

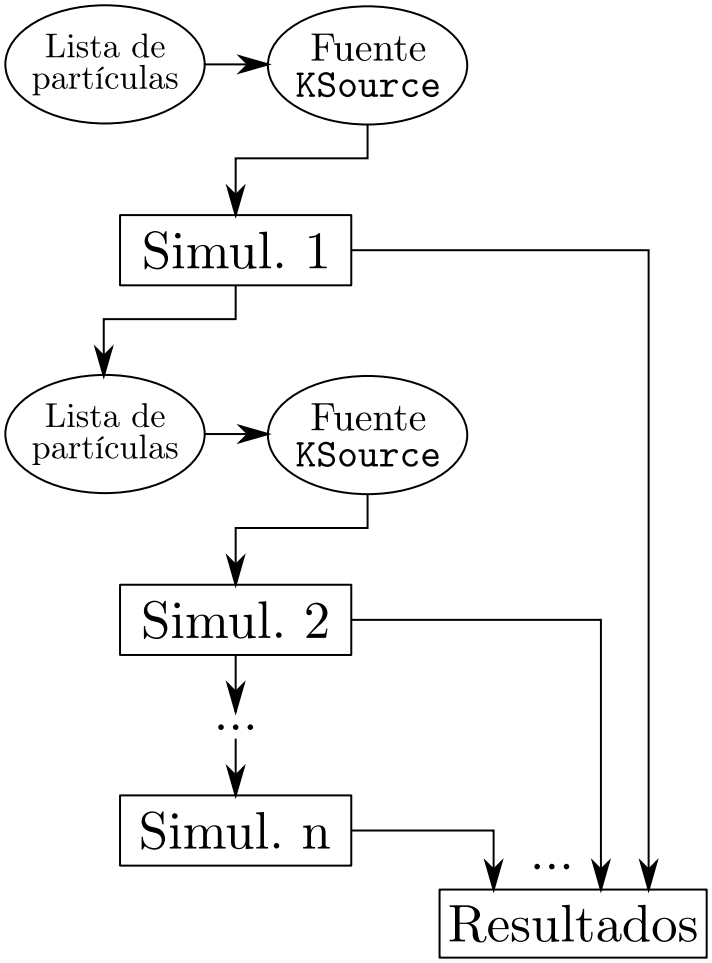


Figura 3.3: Esquema del flujo de trabajo típico.

3.4.1. Simulaciones con McStas y TRIPOLI-4

Si el código utilizado fuera McStas o TRIPOLI-4, se incluyen además archivos plantilla para facilitar las tareas de interpretación de los resultados de las simulaciones. El archivo `postproc.ipynb` permite colectar los principales resultados de una simulación para su registro en una planilla de cálculo. Por otra parte, el archivo `doseplots.ipynb` facilita la creación de gráficos de los mapas de dosis registrados en TRIPOLI-4, utilizando un módulo de Python incluido para tal fin.

En ambos códigos es posible la generación de partículas *on-the-fly*. En McStas esto se logra a través del componente `KDSource.comp`, presente en el subdirectorio `mcstas`. En TRIPOLI-4, por su parte, se debe crear una fuente externa utilizando la plantilla `KDSource.c`, accesible mediante el comando "`kdtool templates --tripoli`". Ambos tipos de fuentes se basan en la utilización de la API en C.

Acople entre óptica neutrónica y transporte con McStas

Entre los componentes de McStas incluidos en el paquete KDSource se encuentra `Guide_shielding`, mediante el cual es posible efectuar cálculos de blindaje en la perfiferia de guías neutrónicas, respetando la física correspondiente a la óptica neutrónica. Esto se logra a través de un acople en un sólo sentido (*one-way coupling*) entre McStas y un código de transporte de radiación, como MCNP, PHITS o TRIPOLI.

El componente `Guide_shielding`, en una simulación de McStas, funciona de igual manera que otros componentes que simulan guías neutrónicas. En particular modela guías de sección rectangular, con posibilidad de curvatura (similar a `Bender`). La diferencia radica en que, al mismo tiempo que propaga los neutrones en su interior, registra en un archivo MCPL los escapes a través de sus espejos, así como las emisiones de fotones *prompt* debidas a absorciones en el sustrato, de acuerdo al estudio presentado en [36]. Es decir que, por cada reflexión, se divide el peso estadístico del neutrón incidente en una fracción reflejada, una transmitida y una absorbida, y se registra en archivos MCPL la porción de neutrón transmitida y los fotones generados por la absorción, con sus energías y pesos estadísticos correspondientes. En McStas la simulación continúa con la porción reflejada.

Las listas de partículas obtenidas representan fuentes superficiales de neutrones escapando a través de los espejos de la guía, y fotones emitidos isotrópicamente. Dichas fuentes tienen forma de tubo, posiblemente curvado, de sección rectangular (ver Fig. 3.2), y son las causantes de la presencia de radiación en la periferia de la guía, la cual debe ser. Los archivos MCPL deben utilizarse entonces para construir fuentes KDSource, con ayuda del archivo plantilla `preproc_tracks.ipynb`, y emplear estas últimas en un cálculo de blindajes de la región que rodea a la guía, típicamente un búnker o *hall*.

Fuentes de activación con TRIPOLI-4 (u otros códigos)

Una fuente de activación es una fuente de radiación gamma proveniente del decaimiento radiactivo de nucleidos inestables presentes en un material, producidos mediante reacciones nucleares (activación neutrónica). Dicha fuente no es superficial, sino volumétrica. Desde luego, la herramienta KDSource es capaz de modelar este tipo de fuentes a través de la técnica KDE, pero, como siempre, necesita para ello una lista de partículas. En este caso dicha lista debe contener las propiedades de los fotones al momento de cada decaimiento, cuya interpretación es la siguiente:

- Energía: Tiene valores discretos, los cuales, junto con sus frecuencias de ocurrencia, conforman el denominado espectro de decaimiento.

- Posición: Se corresponden con los puntos en los que ocurrieron las reacciones de

activación.

- Dirección: Cada gamma de activación tiene una dirección isotrópicamente aleatoria.

Esto quiere decir que, de contarse con una lista de fotones de activación, puede procederse a contruir una fuente KDE con las herramientas descritas anteriormente, en particular con la ayuda del archivo plantilla `preproc_tracks.ipynb`. Debe tenerse la precaución de fijar en cero el escaleo energía, para respetar el carácter discreto del espectro de decaimiento.

Sin embargo, el modo más usual de registrar la activación en materiales es a través de *tallies* volumétricos. Para el código TRIPOLI-4, la API en Python de KDSource cuenta con un módulo que permite convertir de forma sencilla un *tally* de activación a lista de *tracks*, mediante la clase `T4Tally`, utilizando espectros de decaimiento obtenidos del *Live Chart of Nuclides* de IAEA [37]. Además, el archivo plantilla `preproc_tally.ipynb` cuenta con el conjunto de operaciones necesarias para construir una fuente KDE, en base a un *tally* de activación de TRIPOLI-4 y un espectro de decaimiento. Técnicas similares son posibles para otros códigos, aunque no fueron implementadas.

3.5. Verificación

3.5.1. Estimación de densidad sobre muestras con distribución analítica conocida

Para verificar el correcto funcionamiento de la herramienta KDSource, se la utilizó para estimar la densidad de una lista sintética de partículas muestraeada de una distribución conocida. Esto consistió en los siguientes pasos:

1. Elección de la distribución analítica de fuente.
2. Muestreo de una lista sintética de partículas de la distribución de fuente.
3. Ajuste de un modelo KDE sobre la lista de partículas generada.
4. Comparación entre la densidad estimada y la distribcutión analítica.

Todo el proceso de verificación se implementó en un Jupyter Notebook, el cual puede encontrarse en la distribución de KDSource, en el subdirectorio `examples`.

La geometría de fuente elegida fue plana, con la parametrización de variables siendo la letargía u, posición 2D x, y, y un vector de dirección u_x, u_y, u_z. Con respecto a la densidad distribucional de fuente, una elección conveniente podría ser una forma a variables separables, de modo que cada variable pueda ser muestreada de forma independiente.

Sin embargo, se quiso verificar que la herramienta puede modelar adecuadamente una distribución con correlación entre variables. Pero tal distribución puede ser fácilmente lograda mediante la suma de dos distribuciones a variables separables. Por lo tanto, se utilizó la siguiente forma para la densidad distribucional conjunta p:

$$p(u, x, y, z, \mu, \phi) = \left(p_{u,1}(u)p_{x,1}(x) + p_{u,2}(u)p_{x,2}(x)\right) p_y(y)p_z(z)p_\mu(\mu)p_\phi(\phi) \tag{3.4}$$

Donde $p_{u,1}$, $p_{u,2}$, $p_{x,1}$ y $p_{x,2}$ son distribuciones normales, todas con diferentes valores medios. p_y también es una distribución normal, p_μ es lineal entre 0 y 1, y p_ϕ es uniforme (distribución direccional "coseno"), donde $\mu = cos(\theta)$ y ϕ son las coordenadas polares, de modo que el vector dirección es:

$$(u_x, u_y, u_z) = (sin(\theta)cos(\phi), sin(\theta)sin(\phi), cos(\theta)) \tag{3.5}$$

De este modo las variables letargía y x están correlacionadas, pues la distribución presenta dos picos *gaussianos* no alineados en el plano (u, x). Todas las demás variables son independientes. La lista de partículas fue generada con las correspondientes funciones de la biblioteca `numpy`, y fue guardada en un archivo de texto SSV, antes de construir el modelo KDE.

Se utilizó la Divergencia de Kullback-Leibler (KL; ver Cap. 2) para medir la distancia entre la distribución analítica y estimada. En la Figura 3.4a se muestra la KL entre la distribución energética analítica y estimada como función del número de partículas empleado para entrenar al modelo KDE. Se puede ver que, excepto por un ligero ruido estadístico, la KL decrece monótonamente, es decir que cuanto más grande es la lista de partículas empleada, más cerca está la distribución estimada a la real.

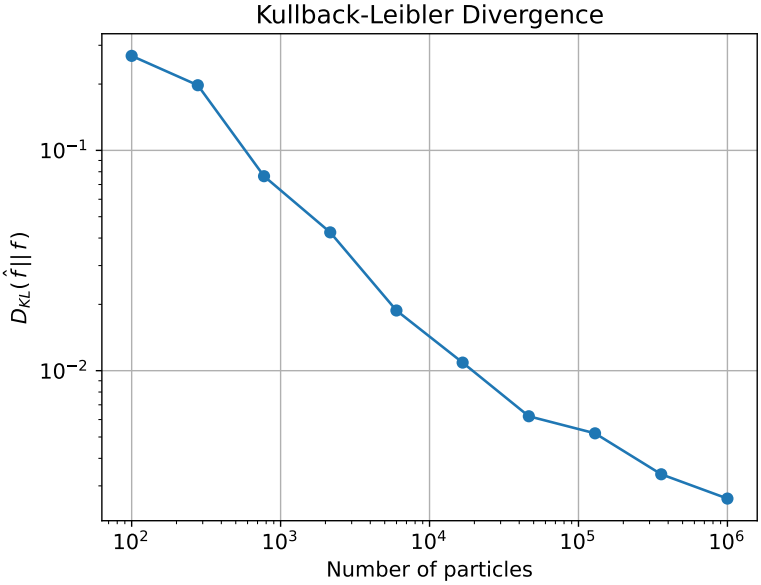
Finalmente se ajustó un modelo sobre una lista de 10^6 partículas, empleando la optimización KNN+MLCV, obteniendo la curva de optimización del factor de mérito del método MLCV (ver Ec. 2.21) que se observa en la Figura 3.5. Se exportó dicho modelo en un archivo XML, y se lo utilizó para generar una nueva lista de partículas, utilizando la aplicación de línea de comando `kdtool resample`. En la Figura 3.4b se compara la distribución energética estimada con el espectro analítico, y se muestra también el histograma de la partículas re-muestreadas, para controlar que el algoritmo de re-muestreo respeta efectivamente la estimación de densidad.

Además, se verificó el adecuado modelado de la correlación entre las distribuciones en energía y x graficando el espectro energético para distintos rangos de x, para ver su variación. En la Figura 3.6a se comparan las distribuciones estimadas y analíticas para $x < 0$ y $x > 0$, y en la Figura 3.6b se muestra la distribución bidimensional estimada en dichas variables. Los valores de KL entre los espectros estimados y real en ambos casos presentaron valores similares a los obtenidos para todo el espectro con $5 \cdot 10^5$

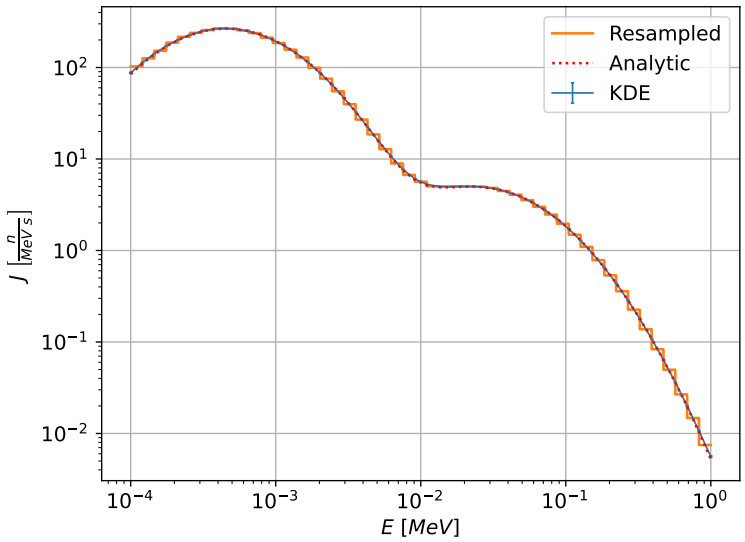
partículas de entrenamiento.

Se verificó también el correcto modelado de las demás variables, como se muestra en la Figura 3.7, para el caso de la componente en z de la dirección. En ese caso, se muestra también la distribución para las partículas con $x > 0$, sólo que al ser u_x una variable independiente de x, esta restricción no modifica la forma de la distribución, sino que solo reduce su intensidad a la mitad.

En esta figura se observa también la diferencia entre la estimación de densidad que se realiza en Python y la empleada por el algoritmo de re-muestreo, correspondiente a



(a) Divergencia de Kullback-Leibler entre la distribución real y la estimada, como función del número de partículas de entrenamiento.



(b) Comparación entre la distribución energética analítica, la distribución estimada y el histograma de partículas re-muestradas.

Figura 3.4: Comparación entre el espectro energético analítico y el estimado

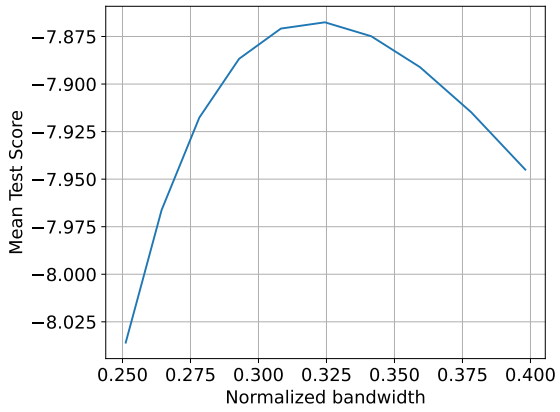
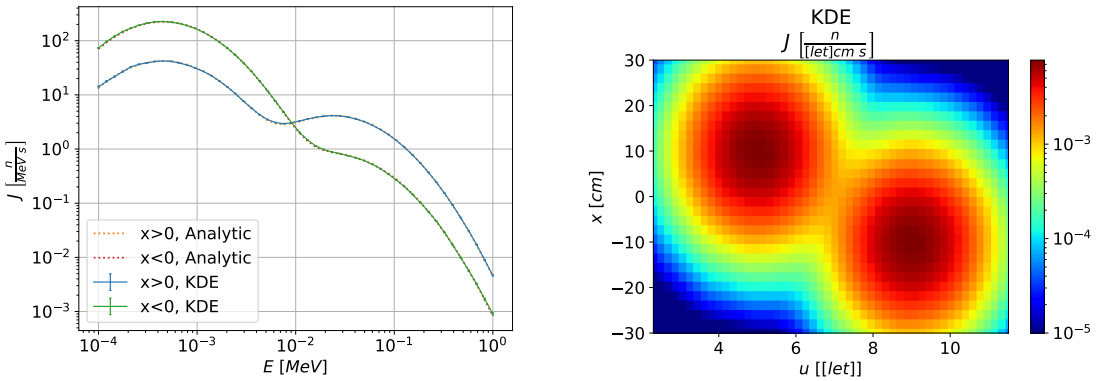


Figura 3.5: Curva del factor de mérito del método MLCV, en función del factor de escaleo para el ancho de banda, con una semilla obtenida por el método KNN.



(a) Comparación entre los espectros energéticos analíticos y estimados para $x < 0$ y $x > 0$.

(b) Distribución bidimensional en el plano $u - x$, donde se observan los dos picos.

Figura 3.6: Verificación del modelado de la correlación entre las distribuciones en energía y x.

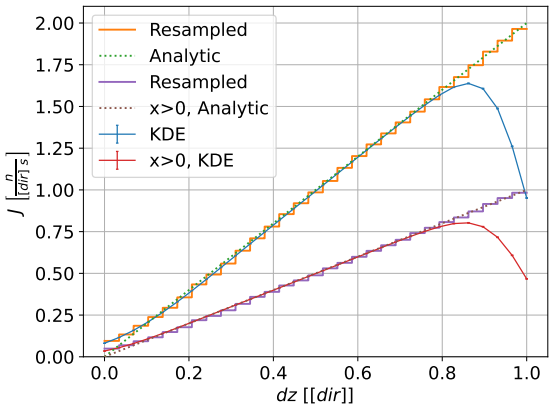


Figura 3.7: Comparación entre la distribución analítica de la componente en z de la dirección, su distribución estimada en Python, y el histograma de las partículas re-muestreadas.

la biblioteca en C. Al haberse empleado una librería externa para el KDE en Python, el modelo no tiene en cuenta características específicas de cada variable, como el hecho de que las componentes de la dirección no pueden ser mayores a 1, lo cual causa un error notable en la estimación cerca de dicho límite. Por el contrario, las funcionalidades de generación de partículas en C sí consideran las particularidades de cada métrica, y gracias a ello se logra respetar el carácter unitario del vector dirección, y modelar adecuadamente la distribución en u_z. Por estos motivos, se recomienda realizar histogramas sobre las partículas re-muestradas para analizar correctamente la distribución estimada, y utilizar las funciones de graficación en Python solo si se conocen sus limitaciones.

3.5.2. Comparación entre KDE y *tracks* para haces

Debido a una de las aplicaciones a las que se orienta KDSource es el modelado de fuentes para haces de neutrones, se provee un módulo para comparación entre KDE y *tracks* en un haz genérico, denominado `beamtest`. Para utilizarlo el usuario debe contar con una fuente KDE ya optimizada, a la entrada del haz. Mediante `beamtest` es posible comparar la corriente a la salida de un colimador utilizando la fuente KDE y la fuente de *tracks*. Ya que lo que se compara es únicamente la corriente integral, es esperable que pueda obtenerse un resultado con error estadístico relativamente bajo incluso con la lista de *tracks*. El módulo `beamtest`, además de simplemente verificar el

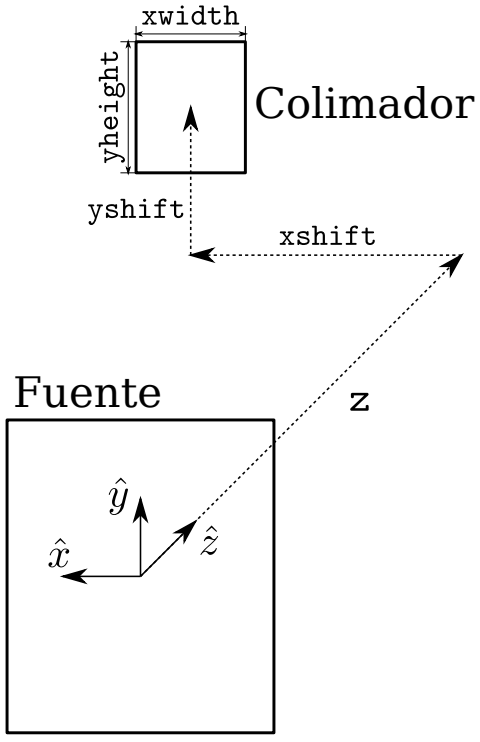


Figura 3.8: Ubicación de la fuente y el colimador en la simulación ejecutada por el módulo `beamtest`.

modelo KDE, puede resultar útil para determinar pequeños ajustes manuales al ancho de banda o al factor de normalización.

El módulo es accesible a través de línea de comando mediante `kdtool beamtest`. A través de argumentos se fija el ancho y altura del colimador (`xwidth`, `yheight`), y la ubicación la fuente (`z`, `xshift`, `yshift`), de acuerdo al esquema de la Figura 3.5.2. La herramienta devuelve las corrientes calculadas con ambas fuentes. Dicho cálculo se realiza en lenguaje C, proveyéndole al módulo independencia del código Monte Carlo a utilizar posteriormente. Con la fuente de *tracks* se simulan todas las partículas de la lista, mientras que con la fuente KDE se elige la cantidad de partículas a generar con el argumento `N`.

Capítulo 4

Cálculo de blindajes en un modelo conceptual de guía de neutrones

Con el objetivo de demostrar la aplicabilidad de la herramienta desarrollada a cálculos de blindajes, se modeló en McStas y TRIPOLI el haz GF1 de RA10. La simulación consta de un acople entre un modelo del interior de la guía en McStas, en el cual se simula la reflexión en los espejos acorde a la óptica neutrónica, y un modelo de los blindajes a su alrededor en TRIPOLI, en el cual se simula el trasporte de radiación y se calculan los mapas de dosis. También se registraron en TRIPOLI mapas de activación, se generaron las fuentes correspondientes, y se calculó la dosis asociada. En esta cadena de cálculo se buscó ejemplificar el procedimiento que debe ejecutarse al utilizar la herramienta KDSource, así como demostrar las ventajas que provee con respecto a técnicas más simples como las fuentes de *tracks*.

El modelo empleado es el mismo que se utilizó en mi Proyecto Integrador [1], simulado además en [2] con McStas, por lo que se buscará corroborar la concordancia de los resultados obtenidos con dichos trabajos. En particular, el principal resultado a comparar es la corriente a la salida de la guía. También se buscó verificar la autoconsistencia de los resultados de cálculos de blindajes, comparando los mapas de dosis con fuentes de *tracks* y fuentes KDE.

4.1. Consideraciones generales sobre los cálculos con fuentes de distribuciones

Las fuentes de *tracks* que dan inicio a la simulación fueron grabadas en una corrida del núcleo y tanque de reflector de RA10, con MCNP6, de tipo criticidad. La cantidad de neutrones generados en el núcleo en dicha simulación fue de $NPS = 10^9$. Según [2] la cantidad de neutrones nacidos por fisión en el núcleo del RA10 será de $FP = 2,3 \cdot 10^{18}\, n/s$, con lo cual la corriente en la superficie de las fuentes de *tracks* se puede

calcular como:

$$J_{fuente} = FP \cdot \frac{I_{fuente}}{NPS} \tag{4.1}$$

Donde J_{fuente} e I_{fuente} pueden referirse tanto a neutrones como a fotones. La misma fórmula es válida si J_{fuente} e I_{fuente} representan densidades, con respecto a energía, posición, ángulo, o conjuntas, es decir que si se multiplican las distribuciones registradas por el factor FP/NPS se obtienen las corrientes en unidades físicas. Si éstas se grafican con las herramientas de la biblioteca `kdsource` en Python, se debe utilizar dicho factor como argumento `fact`.

Cada vez que se utilizan fuentes de distribuciones, la cantidad de partículas nacidas en una superficie es mayor que las que se detectaron en la simulación anterior, lo cual permite mejorar la estadística en las zonas lejanas al núcleo. Por lo tanto, cada vez que se utiliza esta técnica se deberá añadir un factor de la forma:

$$f_{KDSource} = \frac{I^{det}_{KDSource}}{I^{prod}_{KDSource}} \tag{4.2}$$

Donde $I^{det}_{KDSource}$ es la intensidad total detectada en el grabado de la fuente de distribuciones y $I^{prod}_{KDSource}$ es la cantidad de partículas generadas con dicha fuente.

En una simulación compuesta por N corridas, desde el núcleo de RA10 hasta un detector de tipo fuente de distribuciones (`KDSource`), la fórmula general empleada para la conversión de una intensidad registrada en corriente resulta:

$$J = FP \cdot \left(\prod_{i=1}^{N-1} \frac{I^{det}_i}{I^{prod}_{i-1}} \right) \cdot \frac{I}{I^{prod}_{N-1}} = \frac{FP}{NPS} \cdot \left(\prod_{i=1}^{N-1} \frac{I^{det}_i}{I^{prod}_i} \right) \cdot I \tag{4.3}$$

Donde I y J son la intensidad (suma de pesos estadísticos) y la corriente (en partículas/seg), integrales o distribucionales, en la última fuente de detección, I^{det}_i e I^{prod}_i son las intensidades detectadas y producidas por la i-ésima fuente de distribuciones empleada ($I^{prod}_0 = NPS$).

En el caso de la activación neutrónica, ésta es registrada por TRIPOLI con unidades de reacciones en cada celda por partícula de fuente, y convertida luego a fuente de distribuciones de tipo volumétrica. La cantidad registrada resulta ser el cociente I^{det}_i/I^{prod}_{i-1}, siendo I^{det}_i la intensidad de activaciones e I^{prod}_{i-1} la intensidad neutrónica de fuente de la i-ésima corrida. En el equilibrio (estado estacionario), la tasa de activaciones equivale a la tasa de emisiones. En conclusión, incluso empleando tanto fuentes de distribución superficiales como volumétricas, la fórmula 4.3 sigue siendo válida.

Por último, en la última corrida de la simulación se realiza un cálculo de dosis ambiental H*(10), la cual es reportada por TRIPOLI en µSv por partícula de fuente. Por lo tanto, para obtener la dosis en µSv/h se debe multiplicar este valor por la

corriente en partículas por segundo en la fuente de esta región de simulación:

$$H^*(10)\left[\frac{\mu Sv}{h}\right] = 3600\frac{s}{h}\cdot\frac{FP}{NPS}\cdot\left(\prod_{i=1}^{N-1}\frac{I_i^{det}}{I_i^{prod}}\right)\cdot H^*(10)\left[\frac{\mu Sv}{part.gen.}\right] \tag{4.4}$$

El modelo a implementar está compuesto por varios cálculos de dosis, todos calculados con la fórmula 4.4, para obtener por separado las distintas fuentes de dosis en el *hall* de guías. Éstas son:

- Neutrones de fuente
- Fotones de reacciones (n,γ) *prompt*
- Fotones de fuente
- Fotones de activación

Finalmente, la dosis total se calcula como la suma de las dosis parciales debidas a estas fuentes:

$$H^*(10) = H^*(10)_n + H^*(10)_{\gamma f} + H^*(10)_{\gamma p} + H^*(10)_{\gamma a} \tag{4.5}$$

El cálculo de blindajes apunta a la simulación de la operación nominal del reactor, en estado estacionario. En particular, las dosis por activación calculadas corresponden al estado de equilibrio entre generación de nucleidos radiactivos de interés y decaimientos. Es decir que todos los resultados presentados en las siguientes secciones corresponden a dicha condición del reactor. Sin embargo, es posible utilizar los resultados estacionarios para obtener información sobre transitorios, con la metodología que se describe a continuación.

Las tres primeras fuentes de dosis listadas anteriormente pueden considerarse proporcionales a la potencia del reactor instante a instante, ya que sus efectos dinámicos ocurren en escalas de tiempo muy inferiores al segundo. La activación, sin embargo, sí tiene escalas de tiempo apreciables, del orden de minutos o superiores. En las simulaciones se obtiene la distribución de tasa volumétrica de activaciones A la cual, al depender directamente del flujo neutrónico, puede considerarse instante a instante proporcional a la potencia. Por lo tanto, la evolución de la dosis por activación puede obtenerse de la ecuación de conservación de cada nucleido radiactivo relevante:

$$\frac{dN_j}{dt}(t) = A_j(t) - \lambda_j N_j(t) \tag{4.6}$$

$$\Rightarrow N_j(t) = N_j(t=0)e^{-\lambda_j} + \int_0^t A_j(t')e^{-\lambda_j(t-t')}dt' \tag{4.7}$$

Donde λ_j es la constante de decaimiento del nucleido j.

Por lo tanto, puede obtenerse la dosis ambiental a tiempo t, dada una evolución de potencia $P(t) = P_0 f(t)$, en función de las dosis obtenidas para la potencia nominal P_0 en estado estacionario, denotadas $H^*(10)_n$, $H^*(10)_{\gamma f}$, $H^*(10)_{\gamma p}$ y $H^*(10)^{(j)}_{\gamma a}$, siendo j cada unos de los emisores de fotones de activación considerados, mediante la siguiente fórmula:

$$H^*(10)(t) = (H^*(10)_n + H^*(10)_{\gamma f} + H^*(10)_{\gamma p}) \cdot f(t) + \sum_j H^*(10)^{(j)}_{\gamma a} \left(e^{-\lambda_j} + \lambda_j \int_0^t f(t') e^{-\lambda_j (t-t')} dt' \right) \quad (4.8)$$

A modo de ejemplo, en el caso de un apagado brusco del reactor, aproximando una potencia tipo escalón descendente: $P(t) = P_0 \ \forall t < 0$, $P(t) = 0 \ \forall t > 0$, se obtiene la siguiente evolución de dosis ambiental:

$$H^*(10)(t) = \sum_j H^*(10)^{(j)}_{\gamma a} e^{-\lambda_j}, \ \forall t > 0 \quad (4.9)$$

Cabe mencionar que en la presente simulación no se considera la activación dentro del núcleo y tanque de reflector, ya que en dicha zona sólo se realiza una corrida en MCNP, incluyendo sólo neutrones y fotones *prompt*. Esto justifica que los fotones de fuente sean en todo momento proporcionales a la potencia.

4.2. Descripción de los modelos implementados en McStas y TRIPOLI

En el código McStas el modelo se limita al interior de la guía. Inicia en la superficie donde se grabaron las fuentes de *tracks* de las que parte la simulación, ubicada en la entrada a la guía. Las partículas producidas son propagadas por su interior, perdiendo una fracción de su peso estadístico en cada reflexión en un espejo, variable según su energía y ángulo. Las guías están caracterizadas por su valor de m y otros parámetros de la curva de reflectividad [2]. Se utilizó para el modelado de las guías el componente `Guide_shielding.comp` (ver Sección 3.4.1), el cual registra en archivos MCPL la lista de neutrones que escapan a través de los espejos, y la lista de fotones generados por absorciones en el sustrato, las cuales servirán para construir las fuentes en TRIPOLI.

En la Figura 4.1 se muestra un esquema de los tres tramos de los que se compone la guía GF1. En la Tabla 4.1 se muestran las dimensiones de los mismos, así como el valor del m de los espejos. En toda la longitud de la guía la sección transversal es de 7 cm de ancho por 20 cm de alto.

En la Figura 4.2 se presentan algunas imágenes de cortes del modelo implementado

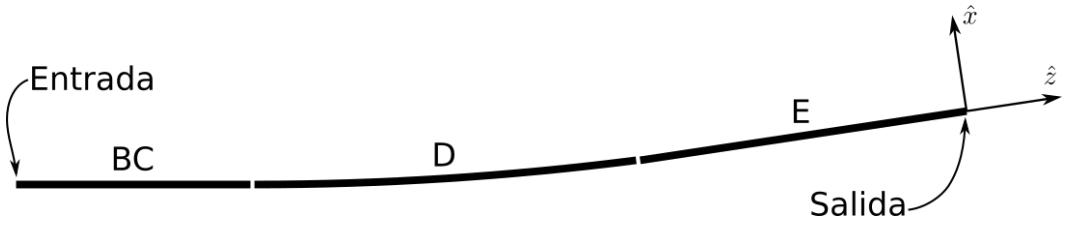


Figura 4.1: Esquema de los tres tramos de los que se compone la guía GF1.

Tramo	Longitud [m]	m espejos				Radio de curvatura [m]
		$x+$	$x-$	$y+$	$y-$	
BC	3	3	3	3	3	-
D	20,2	3	3,5	3	3	929
E	32	3,5	3,5	3	3	-

Tabla 4.1: Dimensiones y propiedades de los espejos en los 3 tramos del haz GF1.

en TRIPOLI, el cual se basa en el diseño del *hall* de guías del RA10, y otras tecnologías típicas de guías de neutrones. Los materiales utilizados se definen en la Tabla 4.2. Las aleaciones de acero y de aluminio, así como el vidrio borado, representan materiales comerciales genéricos de cada tipo, mientras que el hormigón pesado simula un concreto experimental de alta densidad [38]. Cabe mencionar que el modelo realizado, junto con sus materiales, no busca retratar exactamente el *hall* de guías del RA10, sino que consiste en un modelo conceptual con el principal objetivo de demostrar la utilidad de la herramienta realizada.

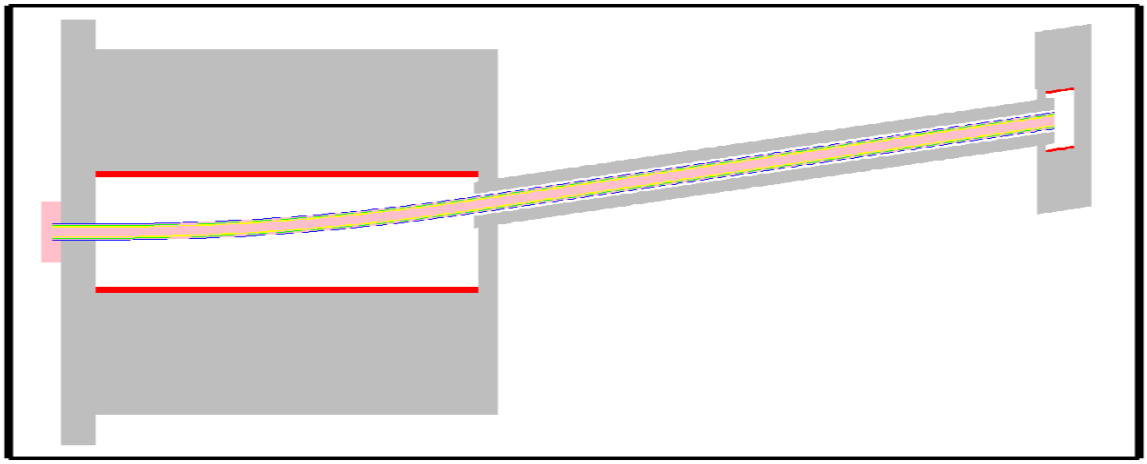
Desde el punto de vista de los componentes simulados, el modelo construido en TRIPOLI consta de los siguientes elementos:

- Guía de neutrones. Ésta se compone por una región interior de sección rectangular rellena con helio a presión subatmosférica, donde se propagan los neutrones, y una serie de capas también de sección rectangular, cada una con espesor constante, las cuales se describen a continuación.

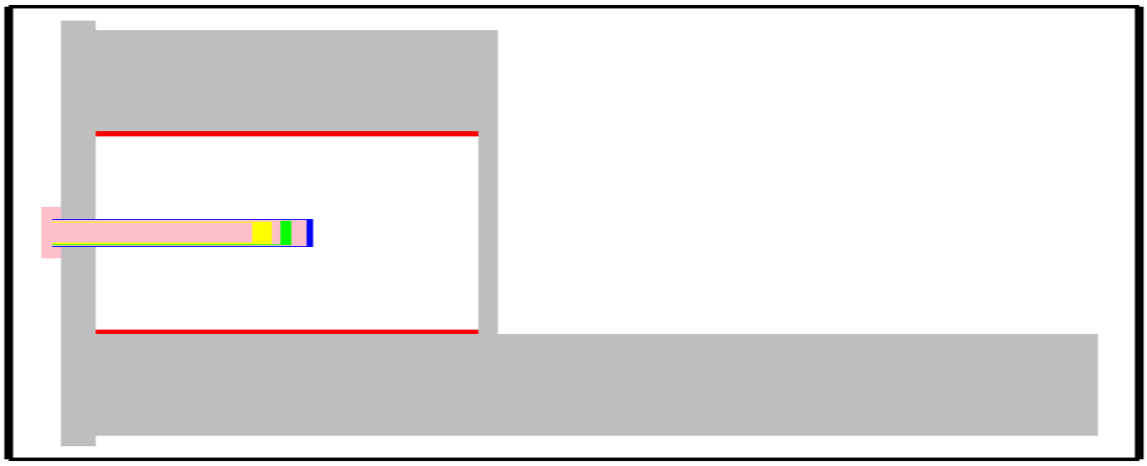
  - Espejos de vidrio borado con un recubrimiento de níquel y titanio en su parte interior.
  - Huelgo de helio.
  - Capa de aluminio, en representación del sistema de alineación de las guías.
  - Huelgo de helio.
  - Sistema de vacío de las guías, compuesto por un tubo de acero inoxidable.

  En la Figura 4.3 se muestra una sección de la guía, generada en TRIPOLI, con las dimensiones de cada parte de la misma.

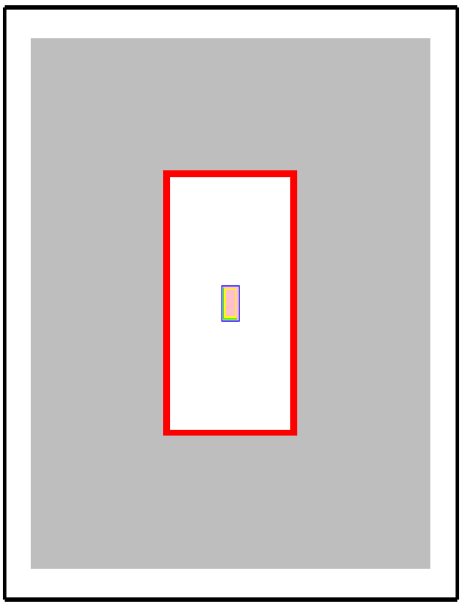
- *Block* del reactor. Consiste en una pared de hormigón pesado, atravesada por el tubo del sistema de vació de las guías. Del lado del reactor sólo se modeló un



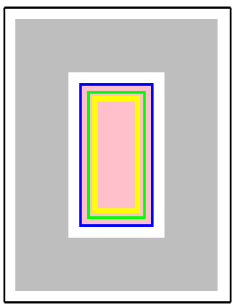
(a) Corte en el plano y=0 (horizontal).



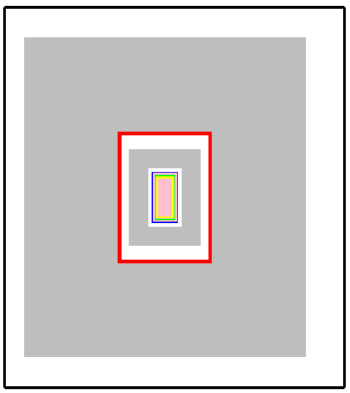
(b) Corte en el plano x=0.



(c) Corte transversal del búnker.



(d) Corte transversal del blindaje exterior.



(e) Corte transversal del *beam catcher*.

Figura 4.2: Distintos cortes del modelo conceptual de búnker y *hall* de guías, implementado en TRIPOLI.

Material	Recubr. de espejos	Vidrio borado	Hormigón pesado	Aleación de aluminio	Acero	Parafina borada
Densidad (g/cm^3)	8,106	2,86	5,87	2,7	8,05	0,9
Elemento/ nucleido	Fracción másica (%)					
H-nat			1,6712			14
B-nat		3,34				5
C-nat					0,15	81
N-nat					0,1	
O-nat		45,11	36,69			
Na-nat		7,61				
Mg-nat				0,9		
Al-nat			0,328889	95,8		
Si-nat		32,42	1,021	1	0,75	
P-nat					0,045	
S-nat					0,03	
K-nat		6,38				
Ca-nat			4,682			
Ti-nat	45,1			0,55		
Cr-nat			15,921	0,25	17	
Mn-nat				0,7	2	
Fe-nat			20,915	0,5	72,925	
Ni-58	54,9					
Ni-nat					7	
Cu-nat				0,1		
Zn-nat				0,2		
W-nat			18,548			

Tabla 4.2: Composición isotópica de los materiales usados en el modelo implementado en TRIPOLI. Los elementos con el sufijo "-nat" representan la suma de todos los isótopos del mismo, con sus abundancias naturales según la información del programa de secciones eficaces GALILEE.

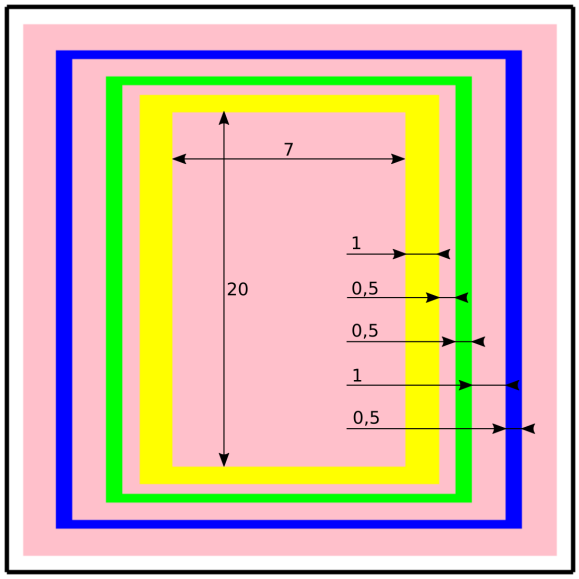


Figura 4.3: Dimensiones del modelo de guías y sistemas asociados, en cm.

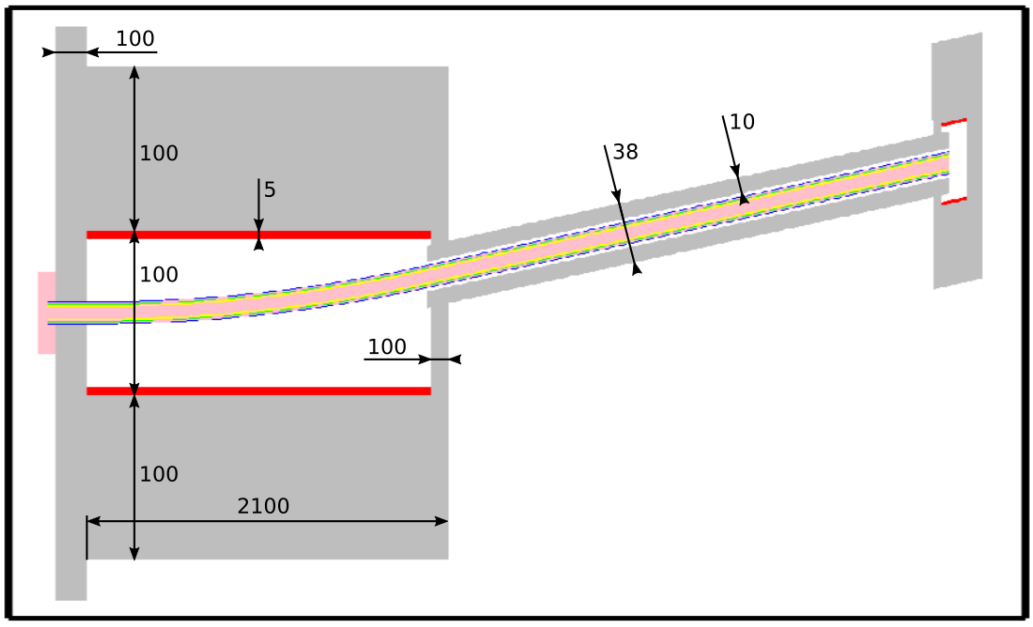


Figura 4.4: Dimensiones de los principales elementos presentes en el *hall* de guías, en cm. En la dirección vertical los espesores de cada blindaje son los mismos, y la altura del interior del búnker es de 200 cm.

volumen de helio rodeando a las estructuras de la guía, representado el tubo de vuelo del haz.

- Búnker de guías. Es un recinto rodeado por paredes de hormigón pesado, con un recubrimiento de parafina borada en su interior. Son 5 paredes: dos laterales, una superior, una inferior y una frontal, las cuales junto con el *block* del reactor encierran un recinto relleno de aire en su composición natural, dentro del cual se encuentra la última parte del primer tramo recto de la guía GF1 y todo el tramo curvo. La pared frontal se ubica inmediatamente después del inicio del segundo tramo recto, siendo atravesada por éste. Las dimensiones del búnker se presentan en la Figura 4.4.

- Blindaje exterior de la guía. En el exterior del búnker, la guía está rodeada por una nueva capa, en este caso de hormigón pesado. Dicho blindaje va desde el inicio del segundo tramo recto de la guía hasta su fin, atravesando la pared frontal del búnker de guías.

- *Beam catcher*. Es una estructura de hormigón pesado, con un recubrimiento interior de parafina borada, que encierra el último tramo de la guía, blindando la radiación causada por la corriente de neutrones térmicos salientes. Desde luego, a la salida de las guías en realidad se ubicarán los instrumentos de experimentación, por lo que el *beam catcher* debe considerarse una simple solución genérica para el blindaje de la salida de las guías.

- *Hall* de guías, es decir el exterior del búnker. Consta de un piso de hormigón pesado que continúa la pared inferior del búnker, aunque sin recubrimiento. Sobre éste se encuentran el búnker de guías, el blindaje exterior y el *beam catcher*, conteniendo a la guía, y sumergidos en un volumen de aire cuyos límites determinan el fin de la simulación. En la Figura 4.4 se presentan las dimensiones de los principales componentes en el *hall*.

4.3. Esquema de la simulación

Desde el punto de las corridas ejecutadas, y de la utilización de fuentes distribucionales, la cadena de cálculo consta de las siguientes etapas, las cuales se esquematizan en la Figura 4.5:

- **McStas: Simulación de neutrones en el interior de la guía**: Los neutrones son producidos con la fuente de distribuciones en una superficie aproximadamente 50 cm antes de la entrada a la guía, y son propagados hasta escapar completamente o alcanzar el final. Se utilizan tres componentes `Guide_shielding.comp` para

los tres tramos de la guía, y un detector plano (`virtual_mclp_output.comp`) a la salida, garantizando que todas las partículas producidas serán registradas al salir del volumen de simulación.

- **TRIPOLI: Simulación del interior del búnker de guías**: Se realizaron tres simulaciones distintas en el interior del búnker:
  - **Fotones de fuente**: En este caso se utiliza la fuente de fotones provenientes del tanque de reflector, y se calcula la dosis que éstos producen en el búnker. Cabe mencionar que en este caso no es necesaria una simulación intermedia en McStas pues no existe ningún fenómeno de óptica para los fotones.
  - **Neutrones de fuente**: Se utiliza la fuente KDE construida con la lista de escapes por los espejos de la guía, registrada en McStas. Se simula tanto la propagación de dichos neutrones como la de los fotones generados por reacciones (n, γ) en esta región. Además de los mapas de dosis por neutrones y fotones *prompt*, se calculan *tallies* de activación en Al-27 y Fe-58.
  - **Fotones de activación**: Con los *tallies* registrados en la simulación de neutrones, y los espectros de decaimiento correspondientes, se generan las fuentes de activación. Se utilizan dichas fuentes como entrada, y se calculan los mapas de dosis.

4.4. Caracterización de las fuentes de *tracks* utilizadas

Como se mencionó anteriormente, las simulaciones realizadas parten de dos fuentes de *tracks* registradas a la entrada de los haces GF1 y GF2 en una corrida de MCNP6, una de neutrones y otra de fotones. En la Figura 4.6 se observa la ubicación de la superficie donde se grabaron dichas fuentes, en el tubo de vuelo de dichos haces, dentro del tanque de reflector del RA10. El formato de las listas de *tracks* empleado fue PTRAC, el cual puede ser convertido a MCPL gracias al *hook* agregado para tal propósito (comando `ptrac2mcpl`).

Dado que en el presente trabajo sólo se busca modelar el haz GF1, los neutrones y fotones registrados en la fuente de *tracks* fueron transportados a la entrada de GF1 con McStas. Dicho transporte se realizó sin ningún método de suavizado o estimación de densidad, simplemente se propagaron los neutrones y fotones en el tubo de vuelo, y se registró en nuevas fuentes de *tracks* aquellos que alcanzaron la superficie de entrada a GF1. Se utilizó para ello el componente `KDSource.comp`, en "modo tracks", es decir tomando las partículas de la lista sin ningún procesamiento. El resultado obtenido es

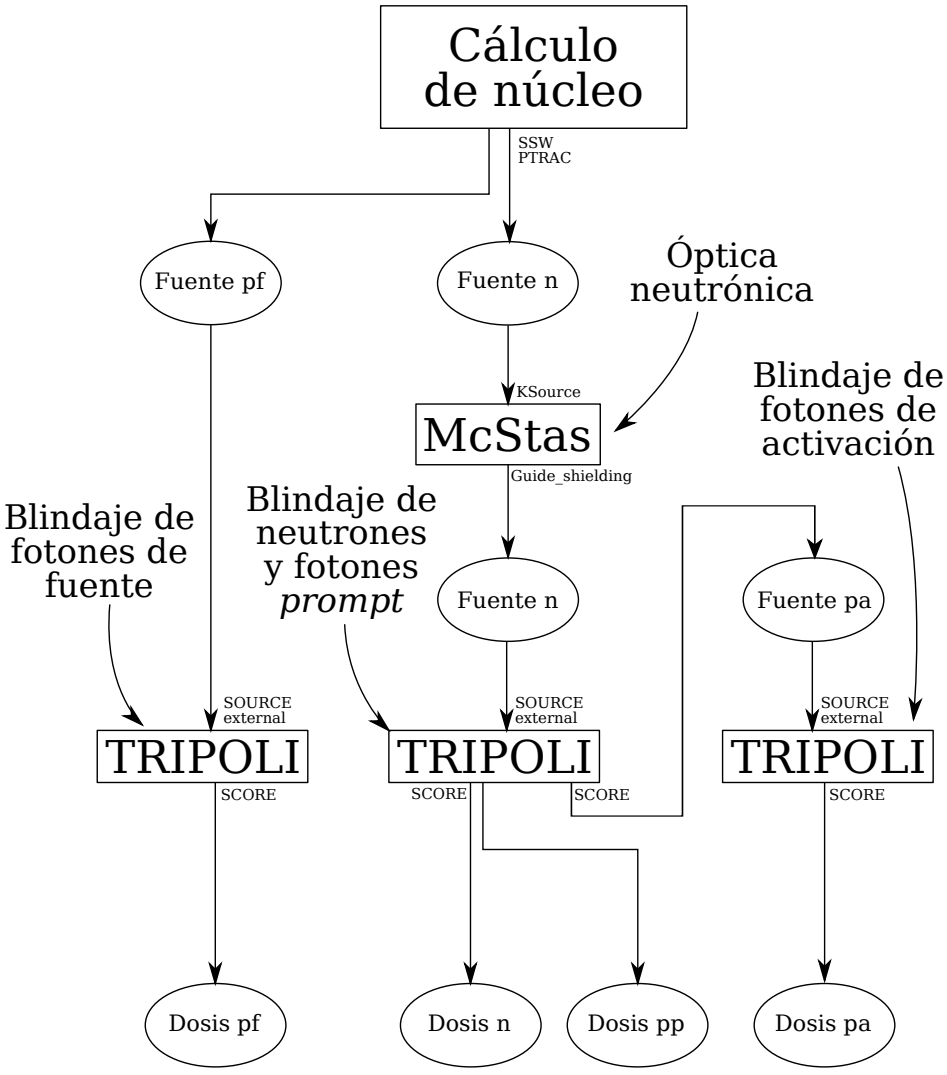


Figura 4.5: Esquema de las simulaciones ejecutadas. n representa neutrones, pp fotones *prompt*, pf fotones de fuente, y pa fotones de activación.

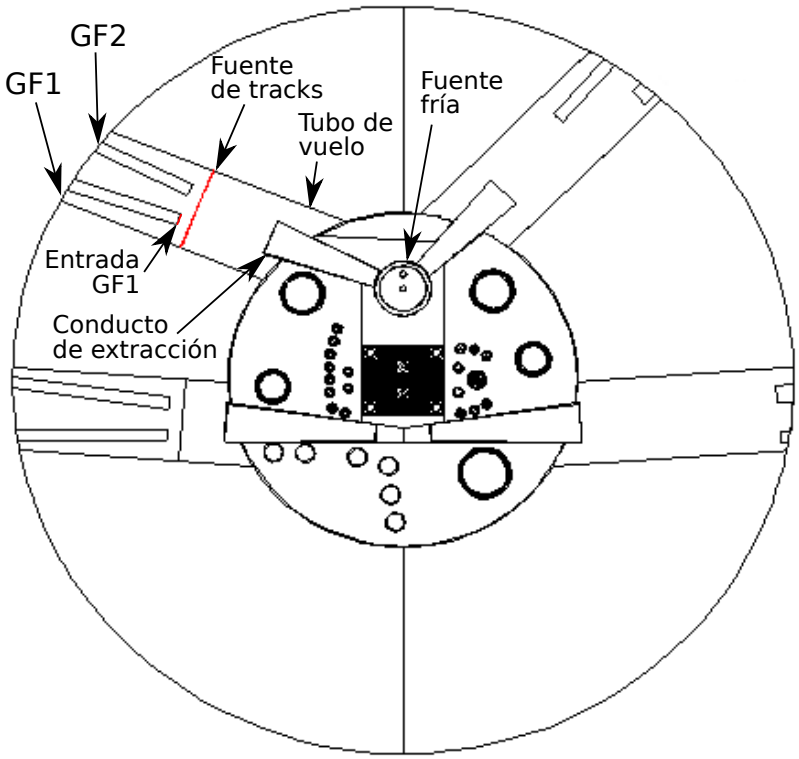


Figura 4.6: Ubicación de la fuente de *tracks* en la geometría empleada para su grabado en MCNP6.

una lista de neutrones y otra de fotones, las cuales se utilizarán como punto de partida para la aplicación del método KDE con la herramienta KDSource. Las partículas que no ingresan a la guía, impactarán en el *block* del reactor, por lo que se espera que no sean relevantes en la dosis en el *búnker* de guías, justificando la decisión de descartarlas.

Para cada lista de partículas se ajustó un modelo KDSource con geometría de fuente plana, tratamiento angular isotrópico, y tratamiento energético en letargía para neutrones, y energía para fotones. Como método de optimización de ancho de banda se utilizó KNN+MLCV, es decir que se empleó el ancho de banda adaptativo obtenido por KNN como semilla para la optimización por validación cruzada. Para KNN se utilizó una cantidad de vecinos de 10, y para el *K-folding* del método MLCV se utilizaron 10 *folds*. Dicho proceso es análogo a lo realizado en la verificación de la herramienta KDSource, descrito en la Subsección 3.5.1.

La cantidad de partículas en las listas de *tracks* a la entrada de la guía fue de $1,813 \cdot 10^5$ neutrones y $4,129 \cdot 10^4$ fotones, con intensidades (suma de pesos estadísticos) de $9,676 \cdot 10^4$ y $5,214 \cdot 10^4$. Considerando la fórmula de conversión a corriente en 4.1,

y el área de la fuente, se obtienen las siguientes corrientes medias:

$$J_n = 1,607 \cdot 10^{12} \ \frac{n}{cm^2 s} \tag{4.10}$$

$$J_\gamma = 8{,}567 \cdot 10^{11} \ \frac{f}{cm^2 s} \tag{4.11}$$

A continuación se presentan gráficos de la fuente de neutrones, incluyendo tanto histogramas de las listas de *tracks* originales como gráficos de las distribuciones de las fuentes KDE ajustadas. Para asegurar que lo que se grafica es efectivamente la distribución de las partículas que se generarán, se registraron en archivos MCPL listas de partículas generadas con las fuentes (resampleadas), y se computaron *tallies* de estas mediante herramientas de Python. En particular se utilizaron las bibliotecas `mcpl` (API para el análisis de archivos MCPL en Python), `numpy` y `matplotlib`.

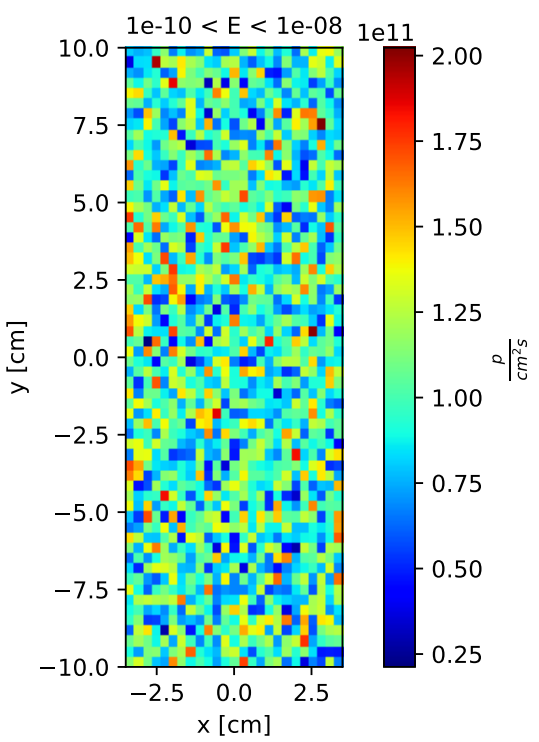
En la Figura 4.7 se muestran histogramas de la distribución espacial de la fuente para 4 grupos de energía. Se observa que las distribuciones son esencialmente uniformes, lo cual se explica teniendo en cuenta el tamaño relativamente pequeño de la fuente, y que la disposición de la fuente fría, el conducto de extracción y la entrada a la guía está diseñada para que la mayoría de las partículas que ingresan provengan de la fuente fría, independientemente de su posición.

Durante el ajuste de la fuente KDE, se tuvo una cierta dificultad para obtener una representación fiel de las distribuciones sobre todas las variables, lo cual es razonable teniendo en cuenta la cantidad relativamente baja de partículas de entrenamiento, y a la multidimensionalidad de la fuente. Sin embargo, teniendo en cuenta la relativa uniformidad de la distribución espacial, y la posibilidad de favorecer la calidad de la estimación sobre ciertas variables (en desmedro de otras), como fue explicado en la Subsección 3.3.4, se favoreció la estimación en energía y en dirección, utilizando los siguientes factores de importancia:

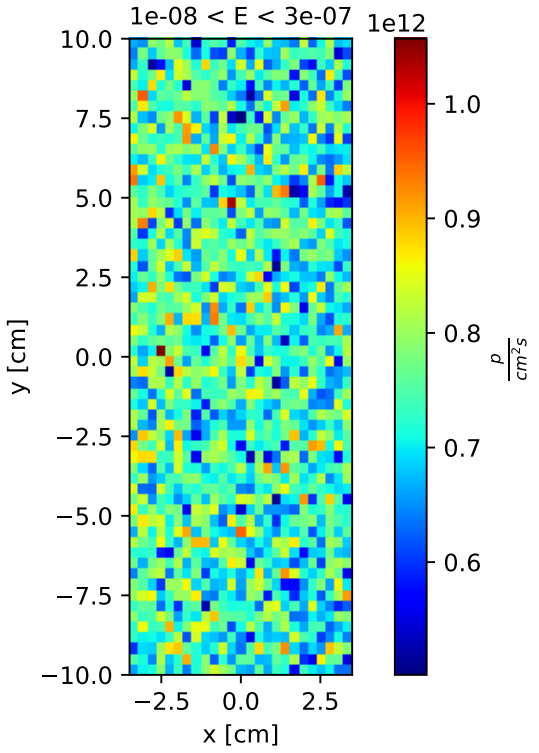
$$\alpha_E = 5$$
$$\alpha_x = \alpha_y = 1$$
$$\alpha_{u_x} = \alpha_{u_y} = \alpha_{u_z} = 10$$

Donde E es la energía, x, y la posición, y u_x, u_y, u_z las tres componentes de la dirección.

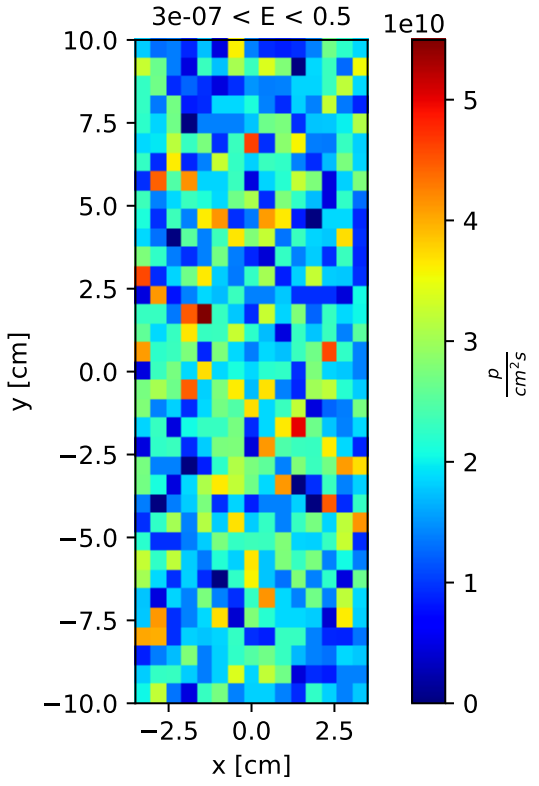
En la Figura 4.8 se muestra el espectro energético de la fuente de neutrones, tanto para la lista de *tracks* como para la fuente KDE. Es posible observar, además de las regiones usuales térmica, epitérmica y rápida, un pico frío, superpuesto con el pico térmico, debido a que los neutrones de esta fuente provienen justamente de una fuente fría. Con respecto a la calidad de la fuente KDE, es posible observar que la fuente KDE coincide con la fuente de *tracks* en la mayor parte del espectro, con una suavidad



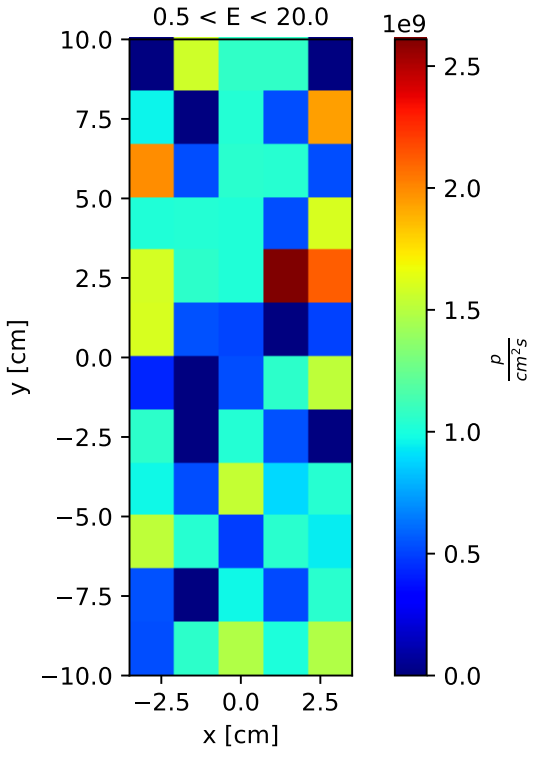
(a) Neutrones fríos.



(b) Neutrones térmicos.



(c) Neutrones epitérmicos.



(d) Neutrones rápidos.

Figura 4.7: Distribución espacial de la corriente de neutrones fríos, térmicos, epitérmicos y rápidos.

notablemente mayor, aunque con un ligero apartamiento en la región de empalme térmico-epitérmico. Para un análisis más preciso, en la Figura 4.9a se muestra el error relativo e_{rel} de la estimación, calculado como:

$$e_{rel} = \frac{J_{KDE} - J_{tracks}}{J_{tracks}} \tag{4.12}$$

Allí se observa notablemente el aumento del error de la estimación, por encima del error estadístico, en la región de empalme térmico-epitérmico. En dicha figura se superpuso además la derivada segunda del espectro en letargía de la fuente de *tracks*, en unidades arbitrarias, para mostrar que las discrepancias entre ambas curvas se deber fundamentalmente al sesgo de suavizado que introduce el método KDE, el cual es proporcional a la derivada segunda de la distribución a estimar (ver Ec. 2.11). Las otras discrepancias que pueden observarse en la región fría y la región rápida también son acompañadas por la derivada segunda de la distribución. Con esto se concluye que el ajuste del espectro resultó satisfactorio, teniendo en cuenta las dificultades esperables de un método de suavizado multidimensional.

En la Figura 4.9b se compara, análogamente, la distribución de la fuente de *tracks* y de la fuente KDE sobre la componente z de la dirección (equivalente a la variable $\mu = cos(\theta)$ de coordenadas polares). En ese caso se observa que el ajuste coincide satisfactoriamente con la distribución original, incluso en la región cercana a $u_z = 1$, donde la derivada segunda de la distribución crece. Esto se debe a las mejoras que aporta el algoritmo de muestreo de la biblioteca en C, que perturba conjuntamente las tres componentes del vector dirección, manteniendo su carácter unitario.

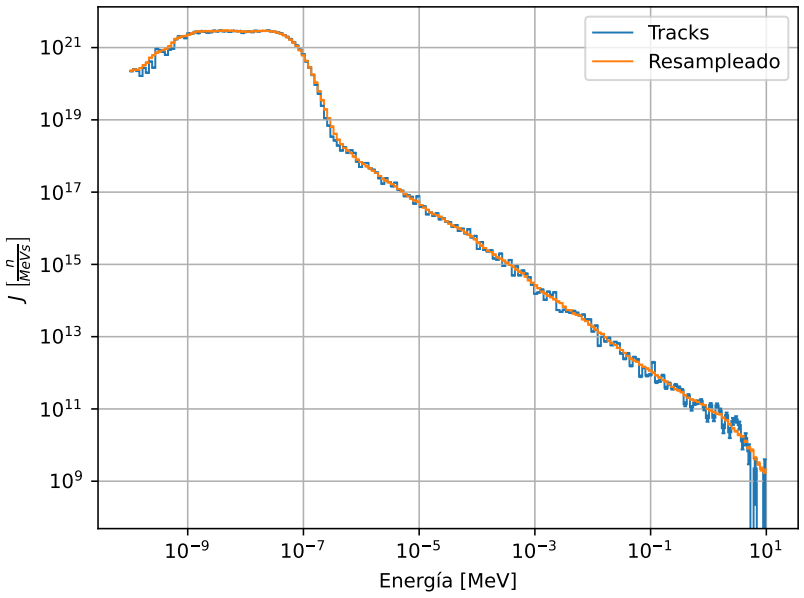
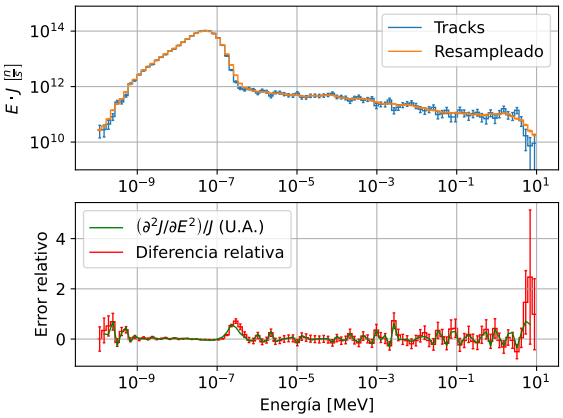
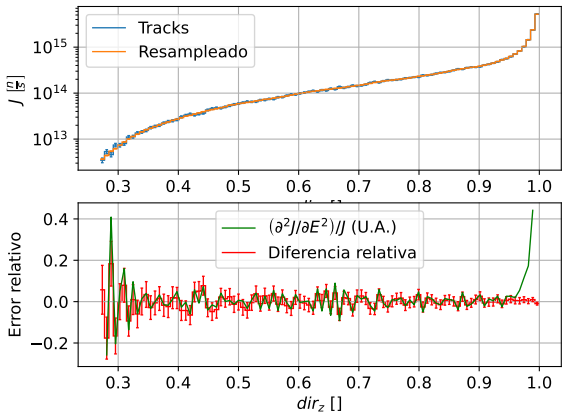


Figura 4.8: Espectro energético neutrónico, para la lista de *tracks* original y la resampleada mediante la fuente KDE.



(a) Comparación entre el espectro energético neutrónico de la fuente de *tracks* y el de la fuente KDE. Abajo se muestra el error relativo de la estimación, y la derivada segunda (discreta) de la distribución de los *tracks*, en unidades arbitrarias.

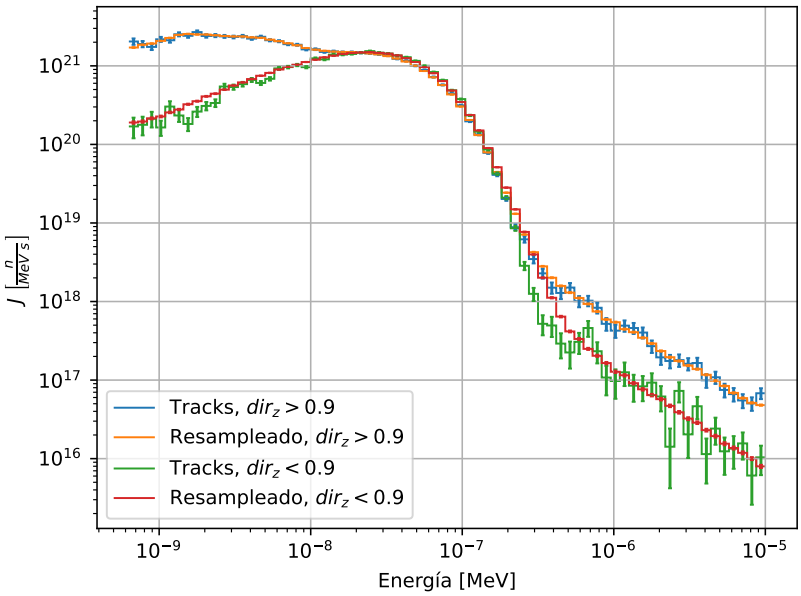


(b) Comparación entre la distribución en u_z neutrónico de la fuente de *tracks* y el de la fuente KDE. Abajo se muestra el error relativo de la estimación, y la derivada segunda (discreta) de la distribución de los *tracks*, en unidades arbitrarias.

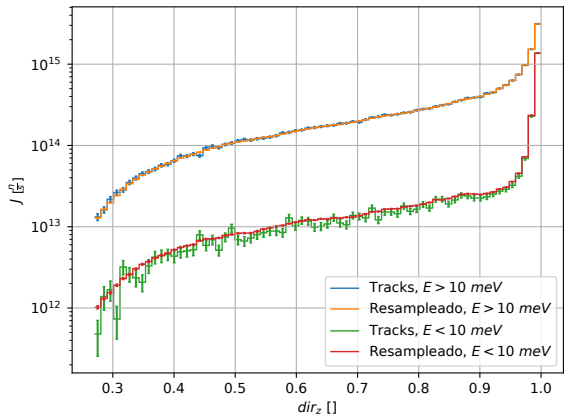
Figura 4.9: Comparación entre las distribuciones en energía y u_z de las fuentes de *tracks* y KDE.

Con respecto a la correlación entre distribuciones, en la Figura 4.10 se comparan ambas fuentes en distintas situaciones que evidencian la correlación entre variables. En la Figura 4.10a se comparan los espectros energéticos de neutrones con $u_z > 0{,}9$ con aquellos con $u_z < 0{,}9$, donde se observa que estos últimos carecen completamente del pico frío, lo cual es razonable considerando que sólo los neutrones con baja divergencia provienen de la fuente fría. En las Figuras 4.10b y 4.10b se compara la distribución direccional de los neutrones fríos con la de aquellos térmicos, epitérmicos y rápidos. El resultado es análogo al del caso anterior, pues se observa que los neutrones fríos están notablemente más concentrados alrededor de $u_z = 1$, lo cual lógicamente implica $u_x = 0$. Finalmente, en las Figuras 4.10d y 4.10e se observa la correlación espacial-direccional de los neutrones fríos. En y se observa lo esperable considerando que la fuente fría se ubica en $y = 0$: la mayor parte de los neutrones fríos en la región superior de la fuente vuelan hacia arriba, y ocurre lo análogo para los la región inferior. En x se observa algo similar, aunque con una ligera desalineación, lo cual probablemente se deba a que la entrada de la guía está alineada con el centro de la fuente fría, pero no con la entrada del tubo de extracción. De estas figuras se concluye que el modelo KDE retrata adecuadamente la correlación entre variables, incluso para x y y, a las cuales se les asignó menor importancia en el ajuste.

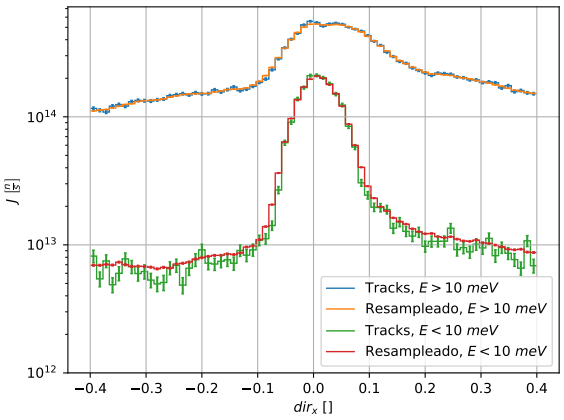
Por último, se evaluó la calidad de la fuente KDE mediante el módulo `beamtest`, descrito en la Subsección 3.5.2, comparando la corriente registrada a través de un colimador con la obtenida con una fuente de *tracks*. Se utilizaron las dimensiones de la sección transversal de la guía para el colimador, es decir 7 cm de ancho por 20 cm de alto. En cada simulación se calculó la intensidad (suma de pesos estadísticos) de



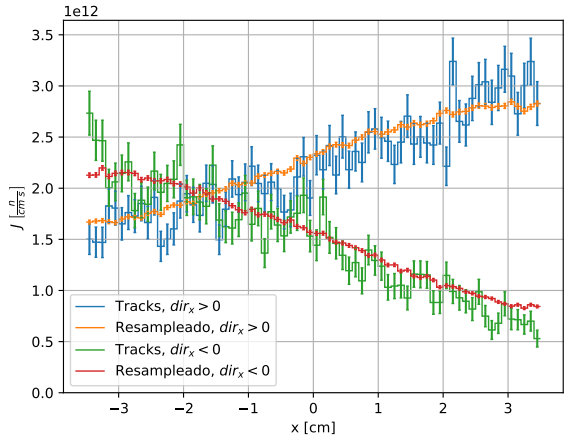
(a) Distribución en energía, variando el rango en u_z.



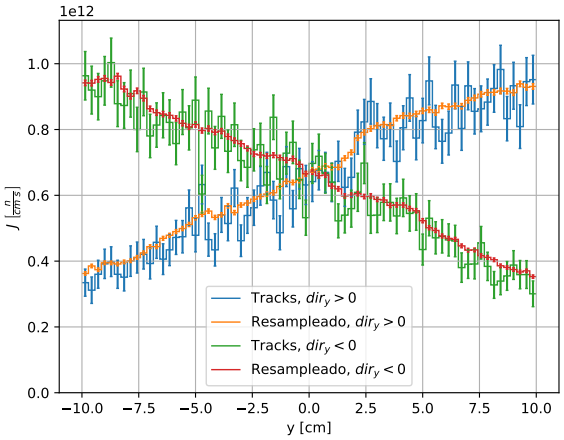
(b) Distribución en u_z, variando el rango en energía.



(c) Distribución en u_x, variando el rango en energía.



(d) Distribución en x, variando el rango en u_x, para neutrones fríos.



(e) Distribución en y, variando el rango en u_y, para neutrones fríos.

Figura 4.10: Comparación entre las correlaciones de las fuentes de *tracks* y KDE de neutrones.

neutrones que entran por el colimador con ambas fuentes, utilizando para la fuente de *tracks* todas las partículas de la lista, y para la fuente KDE 10^6 producciones. Para cada corrida se calculó el error relativo de la fuente KDE ER_{KDE} como:

$$ER_{KDE} = \frac{I_{KDE} - I_{tracks}}{I_{tracks}} \tag{4.13}$$

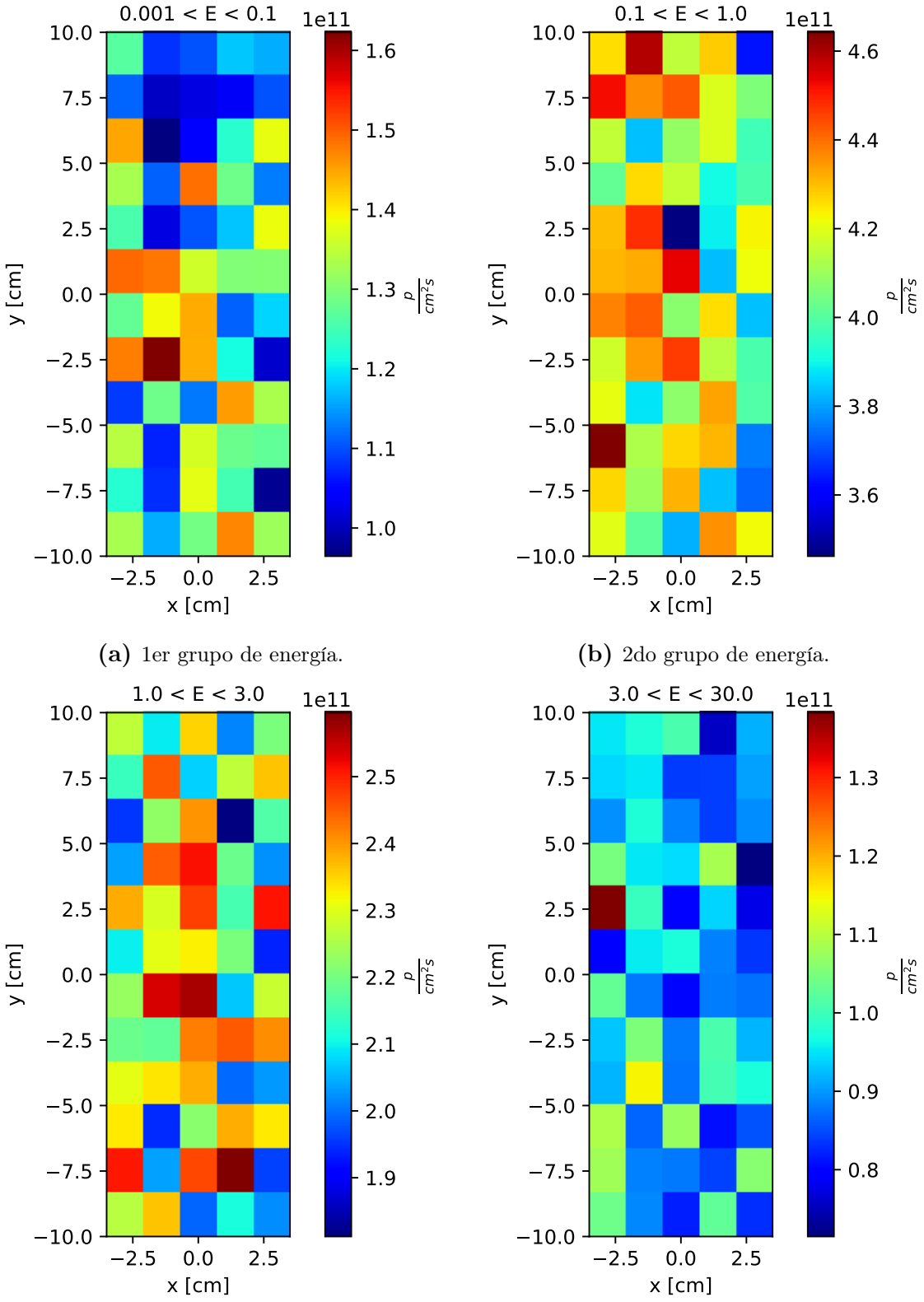
Donde I_{tracks} es la intensidad (suma de pesos estadísticos) registrada en el colimador con la fuente de *tracks*, y I_{KDE} es la intensidad en la misma posición con la fuente KDE. Además, se calculó el error estadístico de ER_{KDE}, y el cociente entre este y su error estadístico.

En la Tabla 4.3 se muestran los resultados obtenidos con el módulo `beamtest` y la fuente de neutrones, para diferentes ubicaciones del colimador (ver Figura 3.8). Se muestra el error relativo de la fuente KDE ER_{KDE}, su error estadístico $\sigma_{ER_{KDE}}$ y el cociente entre ambos. Idealmente, este último cociente debería tener módulo menor a 1, pues esto significa que los resultados con ambas fuentes coinciden, dentro de la banda de error. Se observa que esto ocurre para la mayoría de los puntos evaluados, y que en todos los casos el cociente se mantiene con módulo menor a 3. Las discrepancias se atribuyen principalmente a los efectos de *bias* que introduce el método KDE en posiciones específicas, de forma similar a lo observado en la Figura 4.9a. En etapas posteriores de la cadena de cálculo se realizarán otras verificaciones para controlar que este efecto no sea significativo para los cálculos de blindajes.

`z` [cm]	`xshift` [cm]	`yshift` [cm]	ER_{KDE} [%]	$\sigma_{ER_{KDE}}$ [%]	$\frac{ER_{KDE}}{\sigma_{ER_{KDE}}}$
50	0	0	0.11	0.7	0.16
100	0	0	0.36	1.1	0.33
500	0	0	-9.13	4.1	-2.23
1000	0	0	-7.96	8.1	-0.98
3000	0	0	-23.16	22.8	-1.02
100	-100	0	-6.69	10.1	-0.66
100	100	0	-3.14	8.9	-0.35
100	0	-100	-2.09	9.2	-0.23
100	0	100	9.63	9.5	1.01

Tabla 4.3: Resultados obtenidos con el módulo `beamtest` para la fuente de neutrones a la entrada de la guía. Se muestra el error relativo de la fuente KDE ER_{KDE}, su error estadístico $\sigma_{ER_{KDE}}$, y el cociente entre ambos.

A continuación se presentan gráficos de la fuente de fotones. Para empezar, en la Figura 4.11 se muestran las distribuciones espaciales para 4 grupos de energía. No se observa ninguna tendencia en las distribuciones, aunque es notable la disminución en la estadística con respecto al caso anterior. En la Figura 4.12 se muestra el espectro energético de la fuente de *tracks*. Es posible observar que la distribución no es continua sino que presenta picos, lo cual es esperable en un espectro *gamma*. Por este motivo,



(a) 1er grupo de energía.

(b) 2do grupo de energía.

(c) 3er grupo de energía.

(d) 4to grupo de energía.

Figura 4.11: Distribución espacial de la corriente de fotones para 4 grupos de energía.

al ajustar la fuente KDE, se decidió fijar en cero el factor de *scaling* en energía, lo cual equivale a tomar las energías sin perturbación. Con respecto a los factores de importancia, para la posición y dirección se utilizaron los mismos que para la fuente de neutrones, mientras que para la energía se empleó un factor $\alpha_E = 10^{-3}$ (casi nulo), de acuerdo a lo explicado en la Subsección 3.3.4. Finalmente, en la Figura 4.13 se compara la distribución de ambas fuentes sobre la componente z de la dirección, en la cual no se observa ninguna discrepancia notable.

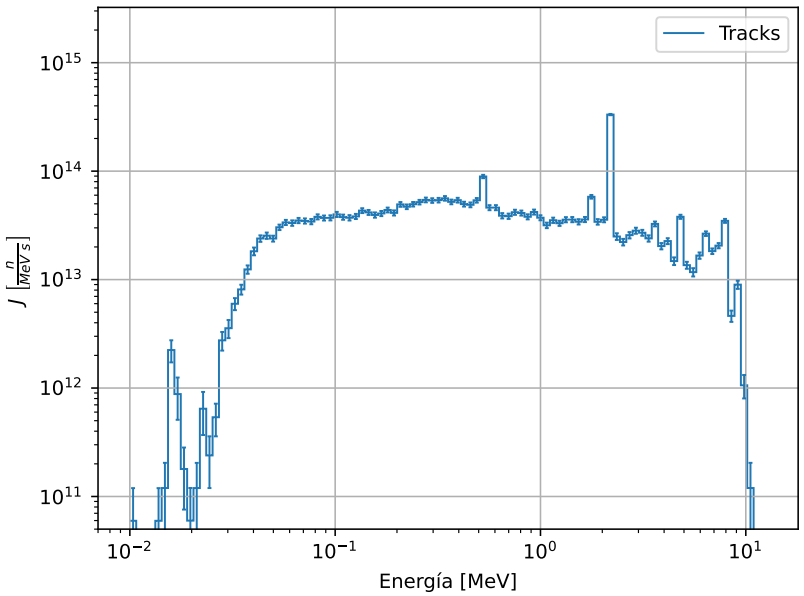


Figura 4.12: Espectro energético fotónico de la fuente de *tracks*.

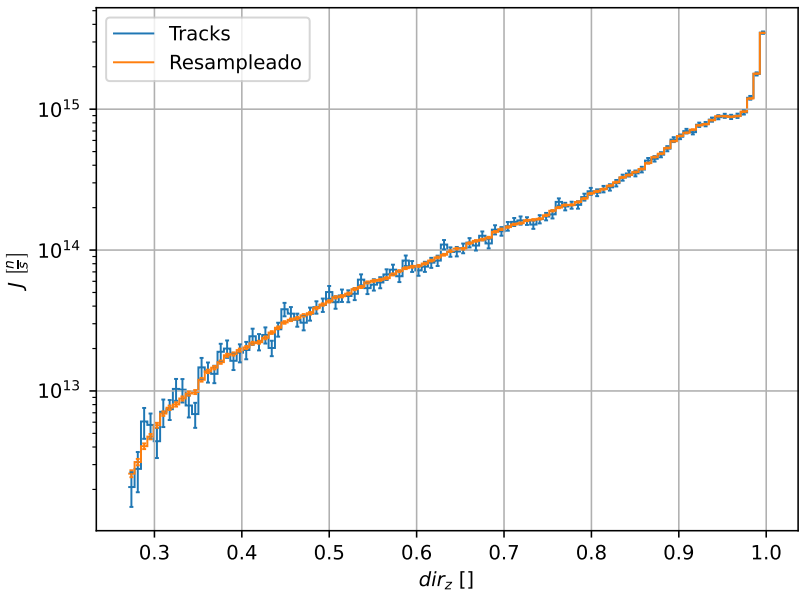


Figura 4.13: Distribución fotónica en u_z, para la lista de *tracks* original y la resampleada mediante la fuente KDE.

4.5. Resultados de las simulaciones

En las siguientes subsecciones se presentan los resultados de las diferentes etapas simuladas, descritas en la Sección 4.3.

4.5.1. Interior de la guía

Se realizó una corrida de 10^8 neutrones, empleando la fuente de distribuciones descrita en la Sección 4.4. Para los tres tramos de la guía se emplearon los componentes guía-detector `Guide_shielding.comp`, dos rectos y uno curvo, con las dimensiones del haz (véase Tabla 4.1). Al final de la guía se empleó una fuente plana, de las dimensiones de la sección transversal, para registrar la distribución de neutrones saliente.

Las listas de neutrones salientes obtenidas en cada tramo se utilizaron para ajustar nuevas fuentes KDE. Las corrientes medias en cada posición se observan en la Tabla 4.4, donde BC, D y E se refieren a los escapes a través de cada tramo de la guía. Se satisfacen las siguientes relaciones, donde A_i refiere a cada área:

$$J_n^{salida} = 0,966 J_{n,tracks}^{salida} = 0,968 J_{n,Fairhurst}^{salida} \tag{4.14}$$

$$J_n^{BC} A_{BC} + J_n^{D} A_D + J_n^{E} A_E + J_n^{salida} A_{salida} = 0,997 J_n^{entrada} A_{entrada} \tag{4.15}$$

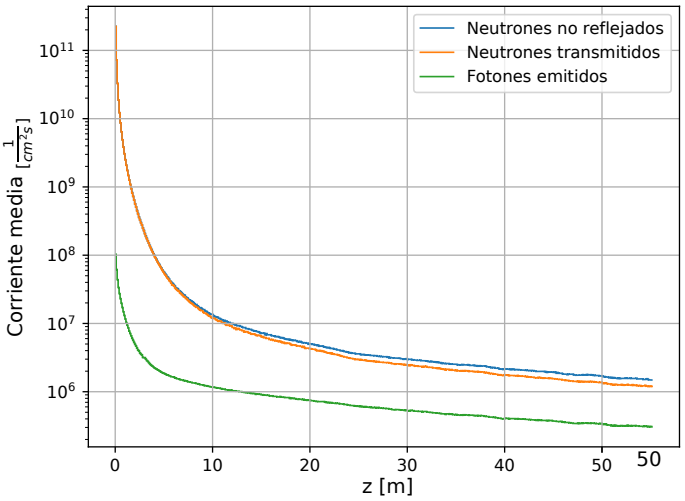
La primera ecuación confirma que la corriente obtenida al emplear la fuente de distribuciones efectivamente reproduce la corriente a la salida de la guía, dentro de un 5 %. De hecho la discrepancia del 3,4 % es incluso menor al error estadístico de la fuente de *tracks*, el cual es del 3,6 %. Además, se observa que los resultados coinciden con los de Fairhurst [2], los cuales también se utilizaron como referencia en mi PI [1]. Las otras dos relaciones verifican el balance de cantidad de partículas, es decir que los neutrones y fotones que ingresan por la entrada o bien escapan por los tramos BC, D y E o bien llegan a la salida. La discrepancia del 0,3 % se debe a escapes en los huelgos de 0,1 mm empleados entre cada par de componentes en McStas, especialmente entre la entrada y el primer tramo de guía.

	Área $[cm^2]$	$J_n \left[\frac{n}{cm^2 s}\right]$ (*tracks*)	$J_n \left[\frac{n}{cm^2 s}\right]$ (KDE)	$J_n \left[\frac{n}{cm^2 s}\right]$ [2]
Entrada	140	$1,589(4) \cdot 10^{12}$	$1,589(4) \cdot 10^{12}$	
BC	16200	$1,347(3) \cdot 10^{10}$	$1,3478(1) \cdot 10^{10}$	
D	109080	$2,25(5) \cdot 10^{7}$	$2,127(2) \cdot 10^{7}$	
E	172800	$1,99(3) \cdot 10^{6}$	$1,9367(9) \cdot 10^{6}$	
Salida	140	$6,4(2) \cdot 10^{9}$	$6,184(8) \cdot 10^{9}$	$6,39 \cdot 10^{9}$

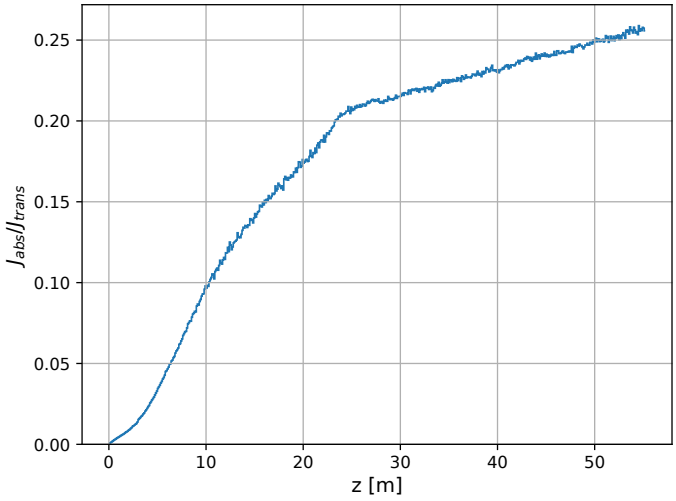
Tabla 4.4: Resultados de las simulaciones del interior de la guía, en McStas, con sus errores estadísticos. Se muestra también el resultado obtenido por Fairhurst en [2].

En la Figura 4.14a se observa la corriente de neutrones escapando a través de la guía en función de la variable z. Como era de esperarse, inicialmente se observa una gran cantidad de escapes, debido a la componente rápida del flujo, así como de neutrones de alta divergencia. A medida que éstos escapan, quedan sólo aquellos de baja energía y ángulo, con lo cual la tasa de escapes disminuye.

En dicha figura se observa además la corriente total de fotones *prompt* emitidos por absorciones en el sustrato de los espejos. En la implementación empleada en mi Proyecto Integrador [1] no se tuvieron en cuenta las absorciones en el sustrato, estudiadas en [36], por lo que se consideraron como escapes la totalidad de los neutrones no reflejados. El efecto práctico de dicha aproximación es que los neutrones que deberían ser absorbidos



(a) Corrientes medias de neutrones no reflejados, transmitidos y fotones emitidos, en función de z.



(b) Cociente entre neutrones absorbidos y neutrones transmitidos, en función de z. El quiebre alrededor de $z = 23\ m$ coincide con el fin del tramo curvo de la guía.

Figura 4.14: Corriente de neutrones escapando de la guía, y fotones emitidos por absorciones en el sustrato de los espejos, en función de z.

en el níquel y titanio del sustrato son, en mayor medida, absorbidos en el boro del vidrio de las guías, debido a que se trata mayormente de neutrones térmicos y fríos. Esto resulta en una subestimación de la dosis alrededor de la guía, ya que la energía media de los *gammas* emitidos por la absorción en níquel y titanio es notablemente superior a la de los resultantes de la absorción en boro, debido principalmente a los fotones de 8,998 MeV y 8,533 MeV del níquel [39]. Como se observa en la Figura 4.14b, este efecto se vuelve más importante a medida que se avanza a lo largo de la guía.

En la Figura 4.15 se observa el espectro energético de los neutrones registrados en los escapes, para distintos valores de z. Allí se observa precisamente el efecto de los espejos, distorsionando el espectro inicial, muy similar al observado en la fuente de *tracks*, en uno mucho más frío, eliminando casi todos los neutrones de más de 100 meV cerca del final de la guía. Se puede observar también que la longitud de la guía no es suficiente para que las fugas lleguen a ser nulas, sino que a pesar del enfriamiento del espectro se siguen registrando escapes hasta el final de ésta.

En la Figura 4.16 se observa la distribución angular de los escapes en el tramo D, para dos rangos de energía y dos regiones distintas en z. Como se esperaba, en todos los casos el pico está cerca de $\mu = 0$ (medido con respecto a la normal de cada espejo), ya que los neutrones con alta divergencia ya han escapado en los primeros centímetros de la guía, pero tampoco exactamente en dicho valor, ya que aquellos con muy baja

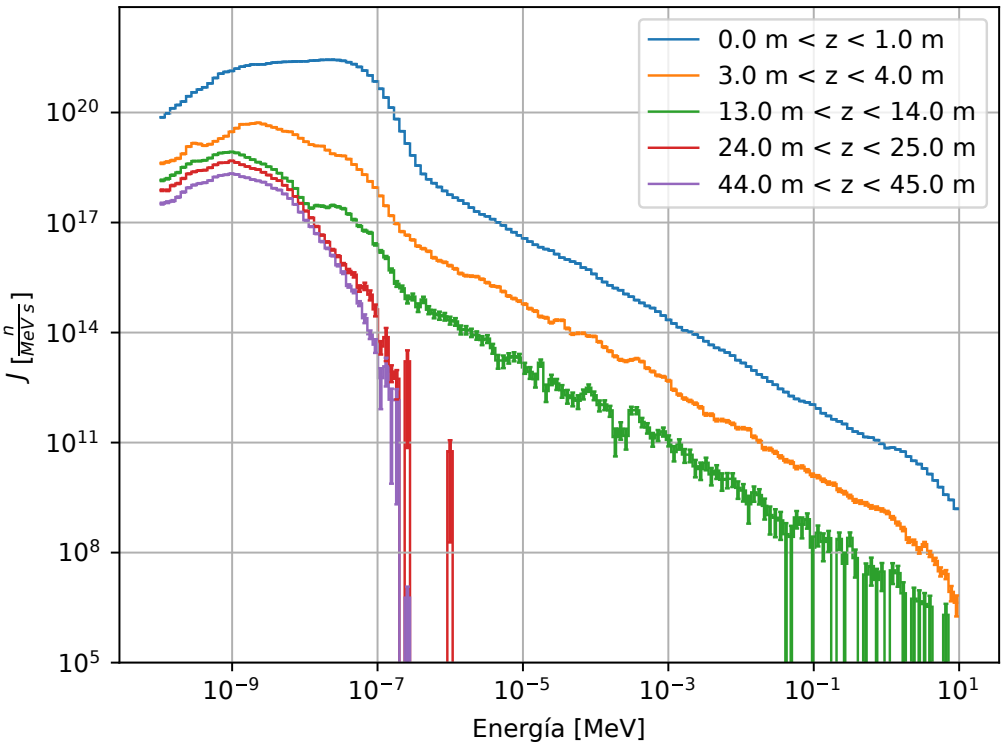
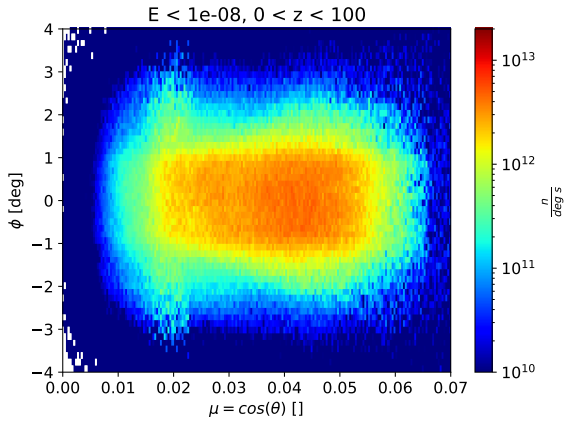


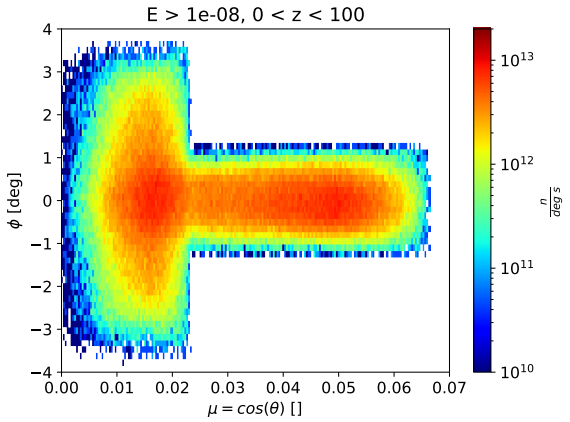
Figura 4.15: Espectro energético de los neutrones escapando de la guía, para varios valores de de z.

divergencia son reflejados. Además, todos los escapes se dan en la región cercana a $\phi = 0$, es decir en la dirección del eje z. Por otra parte, para los neutrones no fríos (energía mayor a 10 meV) se observan cortes notables tanto en μ como en ϕ. Estos se corresponden con los valores límites de dichas variables para neutrones de vuelo directo, considerando el valor de z y las dimensiones de la guía. Para neutrones fríos este efecto es mucho menor por la alta probabilidad de reflexión de los neutrones.

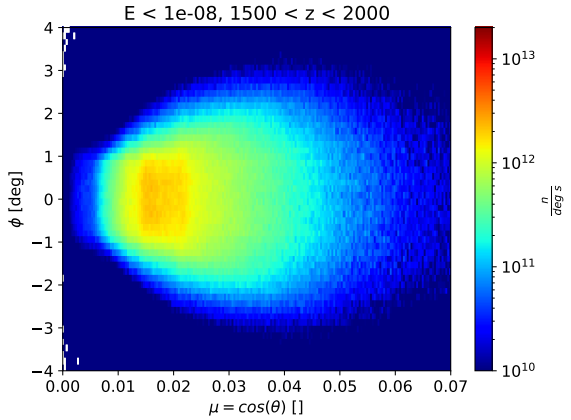
Por otra parte, para observar las ventajas de haber empleado una fuente KDE a la entrada de la guía, en la Figura 4.17 se compara la distribución en función de energía y la componente en z de la dirección con aquella obtenida empleando a la entrada de la guía la fuente de *tracks*. Con esta última fuente apenas es posible apreciar la distribución, mientras que con la fuente KDE se aprecia claramente la dependencia bidimensional. Esto demuestra la utilidad del método KDE en cálculos de haces, permitiendo obtener distribuciones en los puntos de utilización, de forma similar a lo realizado por Fairhurst



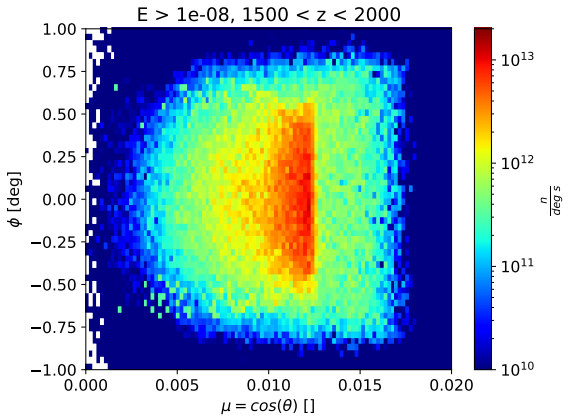
(a) Neutrones fríos, en la región inicial del tramo D.



(b) Neutrones térmicos, epitérmicos y rápidos, en la región inicial del tramo D.

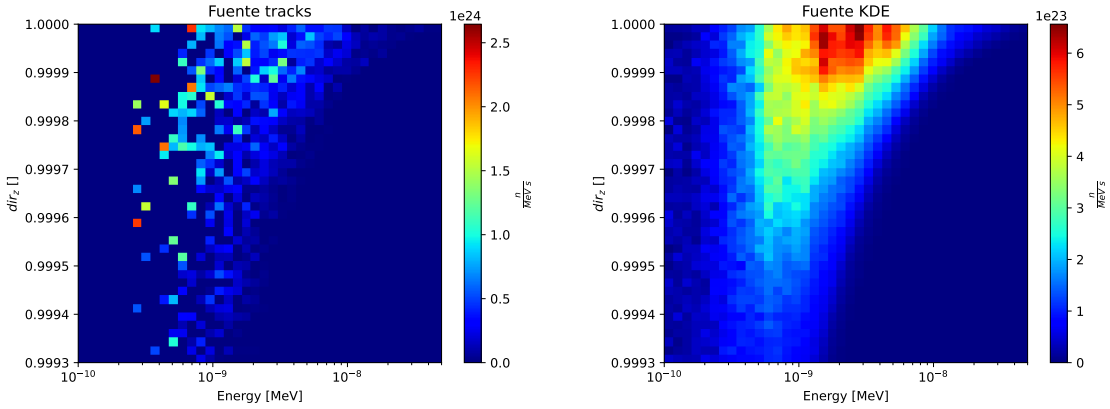


(c) Neutrones fríos, en la región final del tramo D.



(d) Neutrones térmicos, epitérmicos y rápidos, en la región final del tramo D.

Figura 4.16: Distribución angular de neutrones escapando de la guía, para dos rangos de energía y dos rangos en z. Los valores en z son en cm y están medidos con respecto al inicio del tramo D. Las variables polares están medidas con respecto a la normal de cada espejo.



(a) Fuente de *tracks* en la entrada.

(b) Fuente KDE en la entrada.

Figura 4.17: Distribución en función de la energía y u_z a la salida de la guía, para una fuente de *tracks* o una fuente KDE a la entrada.

en [2] con fuentes de histogramas.

Las listas de neutrones escapando de las guías, así como las de fotones emitidos, fueron utilizadas para generar fuentes KDE, las cuales serán utilizadas posteriormente en el cálculo de blindajes alrededor del sistema de guías. Una de las principales dificultades que se observaron fue el efecto de borde que aparece al inicio de cada tramo de la guía, debido a que el método KDE, aún con las funcionalidades diseñadas específicamente para tener en cuenta los límites en el dominio de una fuente, no representa adecuadamente los gradientes en dichas posiciones, y la distribución en función de z presenta justamente allí su derivada máxima (en valor absoluto). Esto se compensó dándole un mayor factor de importancia a la variable z. Como se observa en la Figura 4.18, esto no logró eliminar el efecto de borde, pero sí reducirlo a una pequeña región de unos 20 cm. Los factores de importancia empleados se muestran a continuación, donde se muestra que también se le dio cierta prioridad al espectro energético:

$$\alpha_E = 3$$
$$\alpha_z = 5$$
$$\alpha_t = 1$$
$$\alpha_\mu = 1$$
$$\alpha_\phi = 2$$

Se puede ver que una de las variables tomadas como menos prioritarias es la variables t, es decir la distancia (en cm) recorrida circulando sobre los espejos, a z constante. De todos modos, el algoritmo de perturbación de dicha variable impide que las partículas cambien de espejo, por lo cual el único efecto es un ligero achatamiento de la corriente dentro de cada espejo. En la Figura 4.19 se compara la corriente de *tracks* y

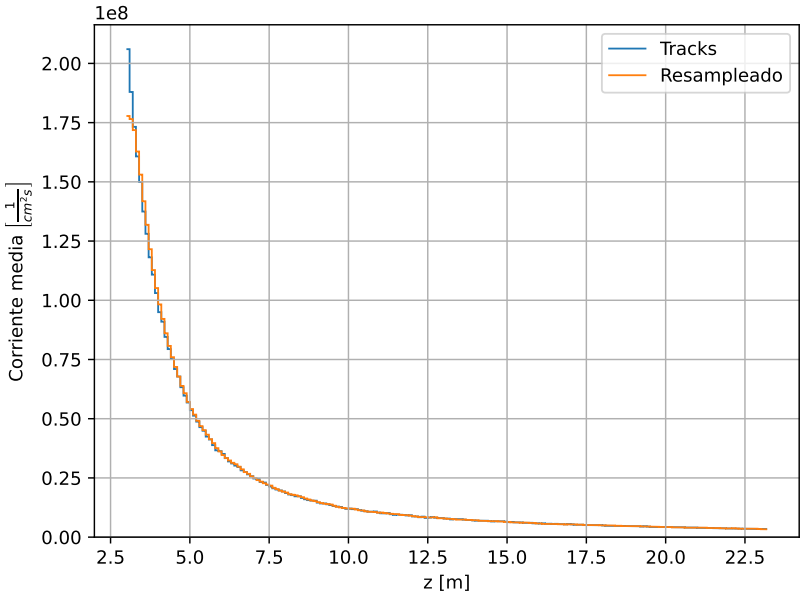


Figura 4.18: Corriente media de neutrones escapando a través de la guía, en función de z, para la fuente de *tracks* y la fuente KDE.

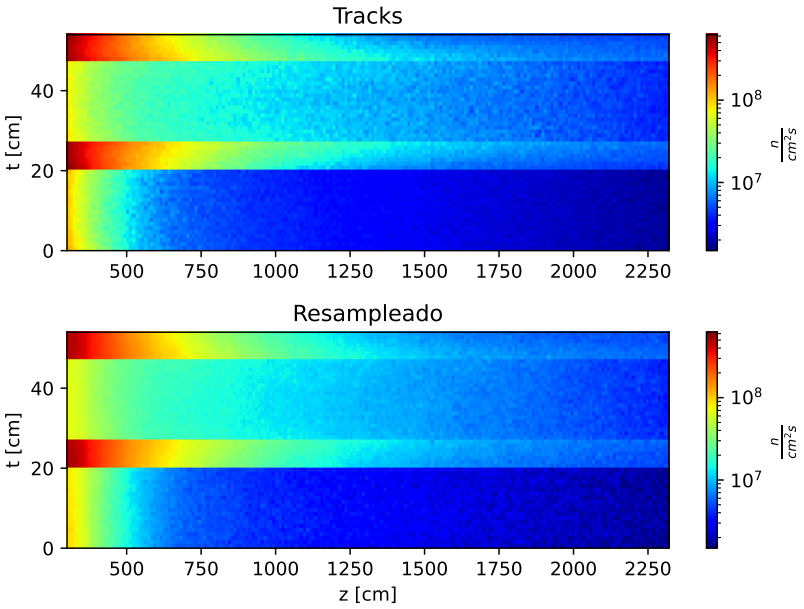


Figura 4.19: Corriente de neutrones escapando a través de la guía, en función de z y la variable de circulación alrededor de la guía, t, para el tramo D.

de la fuente KDE en función de z y la variable t para el tramo D. Para la interpretación de estos gráficos puede imaginarse que los cuatro espejos de la guía fueron desplegados, de forma que puedan ser observados en un mismo plano.

Por otra parte, también se ve relativamente desfavorecida la representación de la distribución angular, como se observa en la Figura 4.20. De todos modos, como se explicará en la siguiente sección, la calidad de la estimación de densidad angular resultó ser suficiente para que los mapas de dosis obtenidos en la periferia de la guía sean

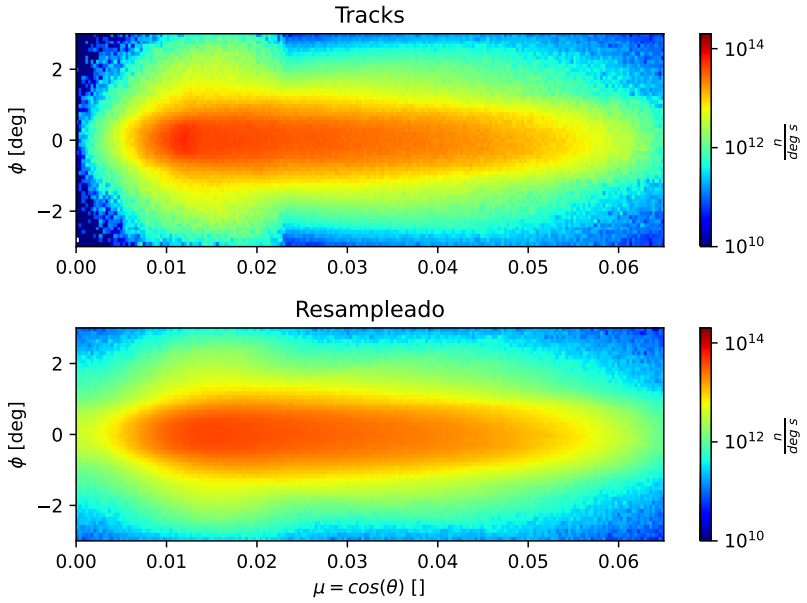


Figura 4.20: Distribución angular de los neutrones escapando a través de la guía, en función de las variables polares medidas con respecto a la normal de cada espejo.

consistentes con los obtenidos con la fuente de *tracks*. Cabe mencionar que se verificó que los principales fenómenos observados en la Figura 4.16, con respecto a la correlación entre la distribución angular y la energía y posición, siguen observándose, aunque con cierto suavizado en cada caso.

Durante el ajuste de la fuente KDE de fotones emitidos por absorciones en el sustrato de las guías se emplearon factores de importancia de 10^{-3} (casi nulos), para la energía y la dirección, ya que luego de la optimización se fijó en cero el *scaling* en energía, y en infinito los de dirección. De este modo se logra representar exactamente el espectro de emisión del níquel y titanio, y se tiene una emisión isotrópica. Además esto simplificó notablemente el correcto modelado espacial, ya que el efecto práctico es una reducción de la dimensionalidad de la fuente de 5 (energía, posición z, t y dirección μ, ϕ) a 2 (solo posición z, t).

4.5.2. Búnker de guías

Antes de utilizar la fuente de neutrones provenientes del interior de la guía, y efectuar correctamente el acople óptica-blindaje, se decidió observar los mapas de dosis en el búnker empleando la fuente a la entrada de las guías. Además, únicamente para la lista de *tracks* de neutrones, se compararon tres fuentes diferentes:

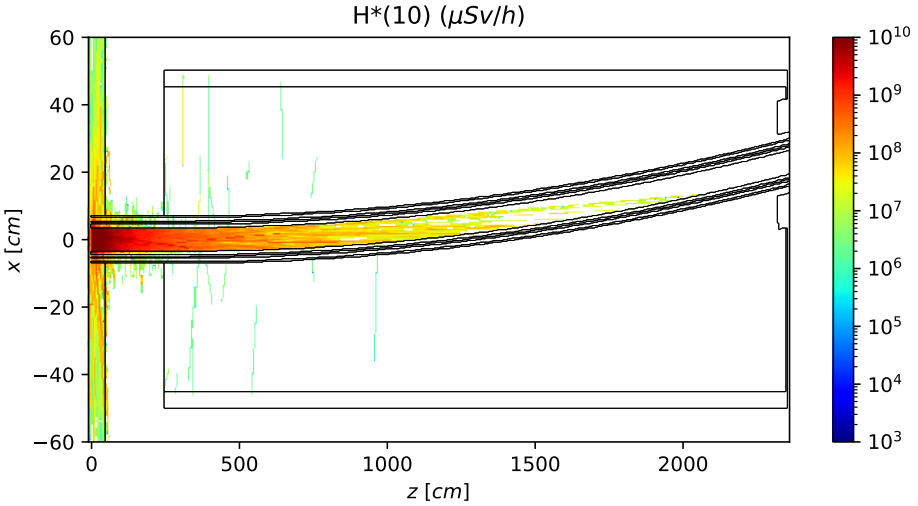
- Fuente de *tracks*, sin repetición de las partículas.
- Fuente de *tracks*, con repetición de las partículas.
- Fuente KDE.

En las Figuras 4.21 y 4.22 se muestran los mapas de dosis por neutrones y fotones *prompt*, respectivamente, registrados al utilizar los tres tipos de fuentes. Para la fuente de *tracks* sin repeticiones el número de partículas de fuente equivale a la longitud de la lista, es decir $1,813 \cdot 10^5$. Con las otras dos fuentes se generaron 10^9 partículas. Cabe mencionar que la fuente de *tracks*, al tomar las partículas directamente de la lista, toma los pesos también de allí, mientras que la fuente KDE los normaliza a 1, como se explica en la documentación de la función de muestreo `KDS_sample` (ver Sección A.4). En todos los casos la dosis está promediada en $-5\ cm < y < 5\ cm$.

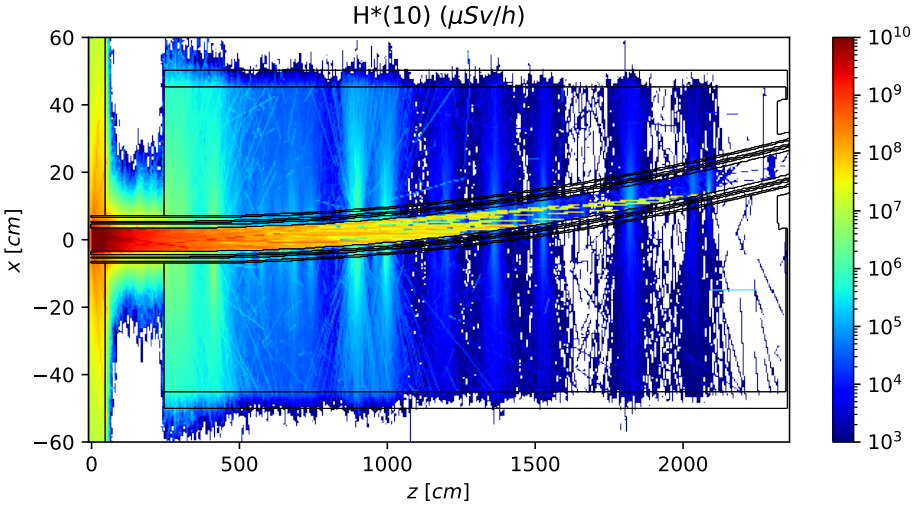
Se puede observar que para la primera fuente la estadística es muy pobre, insuficiente para apreciar adecuadamente la distribución de dosis. Al agregar repetición a la fuente de *tracks*, la estadística mejora. Sin embargo, aparecen falsas "zonas calientes" en ciertos puntos de la guía, las cuales se deben a la estadística limitada de la fuente. En el interior de la guía los neutrones se propagan en vacío, por lo que repetir el muestreo de una misma partícula lleva a la misma trayectoria, causando la aparición de "rayos" de alta dosis, y los correspondientes picos en el punto donde la trayectoria choca contra una pared de la guía. La fuente KDE aumenta notablemente la suavidad del mapa de dosis, logrando incluso que este cubra una mayor región del búnker. Los resultados obtenidos coinciden con el estudio teórico del error asociado a las fuentes provisto en el Apéndice B, en particular para problemas de propagación en vacío. Por último, en la Figura 4.23 se comparan las dosis sobre un corte longitudinal, sobre $x = -10\ cm$ (promediada en un intervalo de 25 mm), y se puede observar que, en las regiones en que la estadística permite una comparación, las curvas coinciden, lo cual significa que el *bias* presente en la fuente KDE no es significativo.

De todos modos, más allá de los diferentes tipos de fuentes, las simulaciones recién presentadas no respetan la física de los neutrones en el interior de la guía, pues TRIPOLI no permite el modelado de espejos neutrónicos. Para ello se debe acoplar la simulación con la corrida realizada en McStas, lo cual consiste en emplear para la construcción de las fuentes las listas de escapes registradas en McStas en la simulación presentada en la sección anterior. Para comparar las ventajas que aporta el método KDE, además de utilizar una fuente KDE con dicha lista, se utilizaron fuentes de *tracks* que emplean la lista de escapes obtenida en McStas al utilizar la fuente de *tracks* a la entrada de la guía, con y sin repetición.

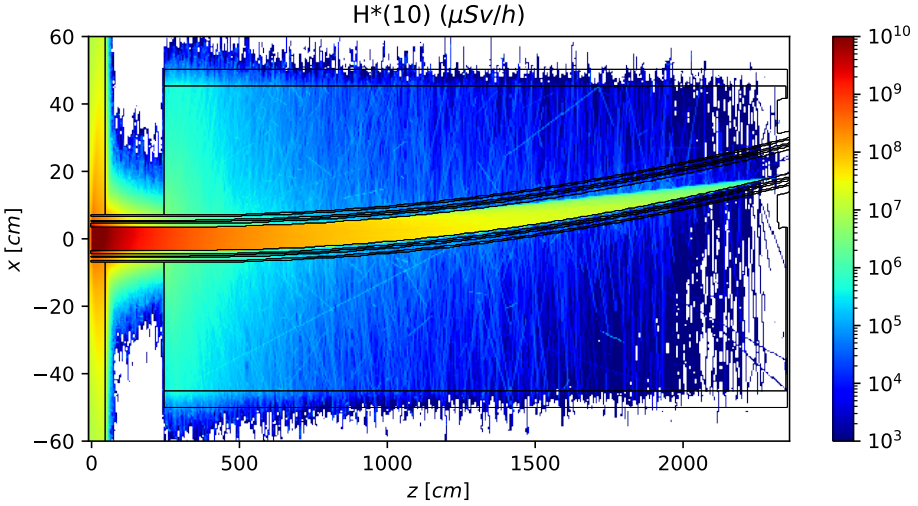
En las Figuras 4.24 y 4.25 se muestran los mapas de dosis por neutrones y fotones *prompt*, respectivamente, registrados con los tres tipos de fuentes. En los sorteos de neutrones escapando de las guías, implementados con la plantilla de fuente externa de TRIPOLI `KDSource.c`, se utilizó la técnica de *source biasing* para favorecer la producción de partículas en el tramo D, el más significativo para esta etapa de la simulación. En cada una de las tres simulaciones se produjeron 10^9 partículas, y los mapas de dosis están promediados en $-5\ cm < y < 5\ cm$.



(a) Fuente de *tracks* sin repetición.

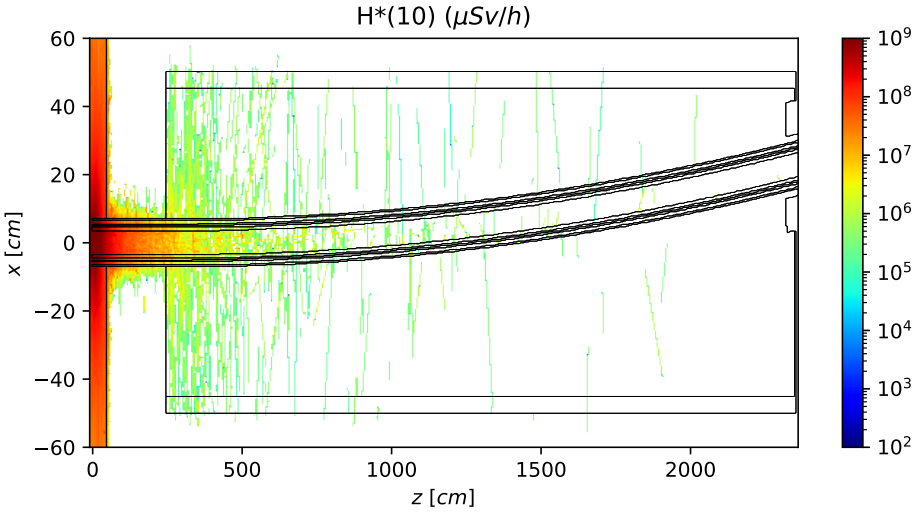


(b) Fuente de *tracks* con repetición.

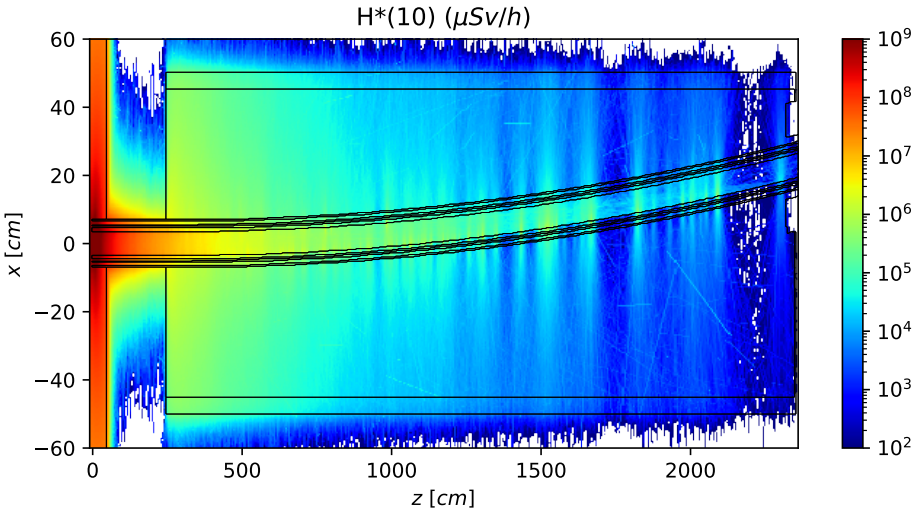


(c) Fuente KDE.

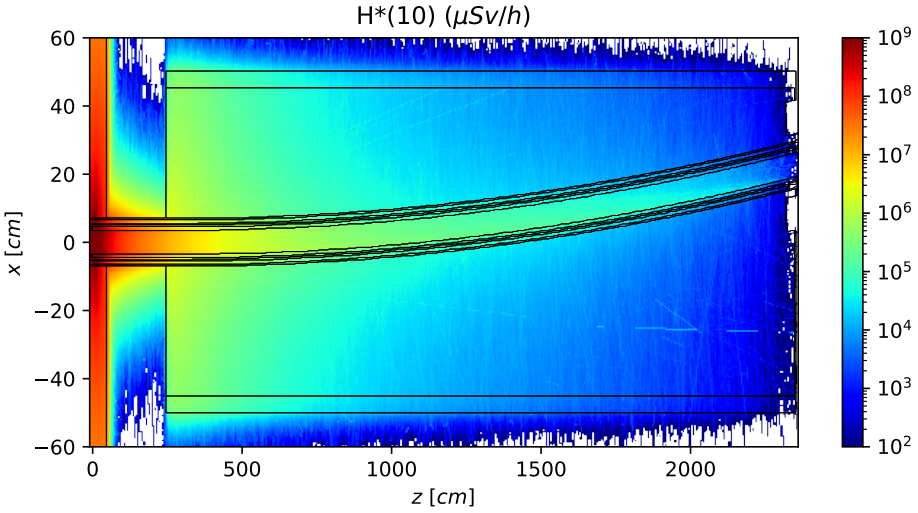
Figura 4.21: Mapas de los dosis por neutrones, con fuentes a la entrada de la guía.



(a) Fuente de *tracks* sin repetición.

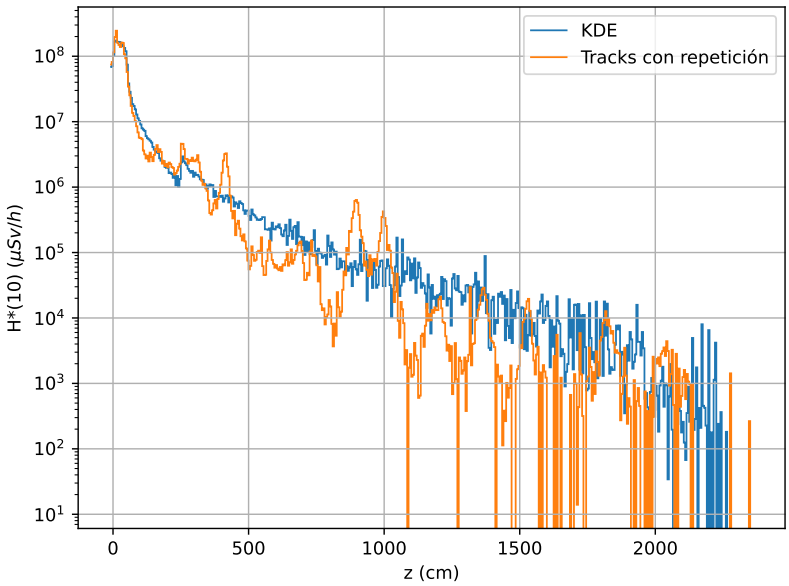


(b) Fuente de *tracks* con repetición.

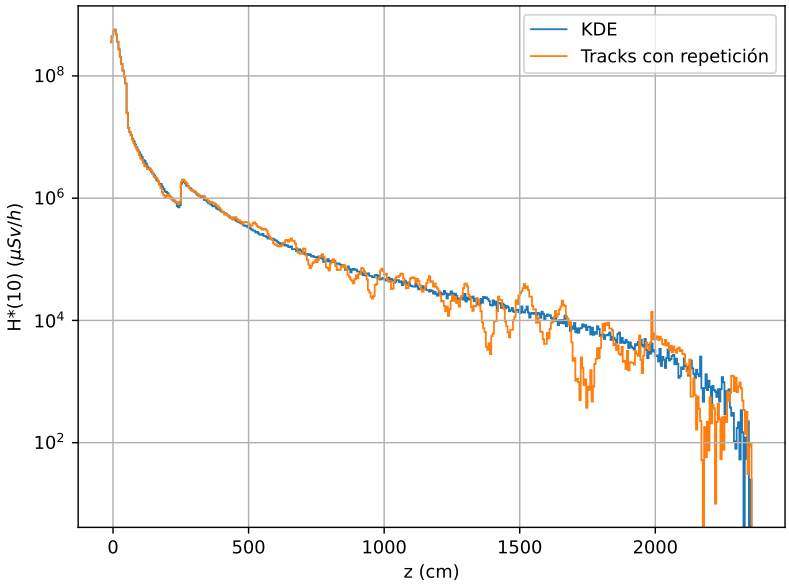


(c) Fuente KDE.

Figura 4.22: Mapas de los dosis por fotones *prompt*, con fuentes a la entrada de la guía.



(a) Dosis por neutrones.



(b) Dosis por fotones *prompt*.

Figura 4.23: Dosis en función de z, sobre $x = -10$ cm, para fuentes de *tracks* con repetición y KDE, a la entrada de la guía.

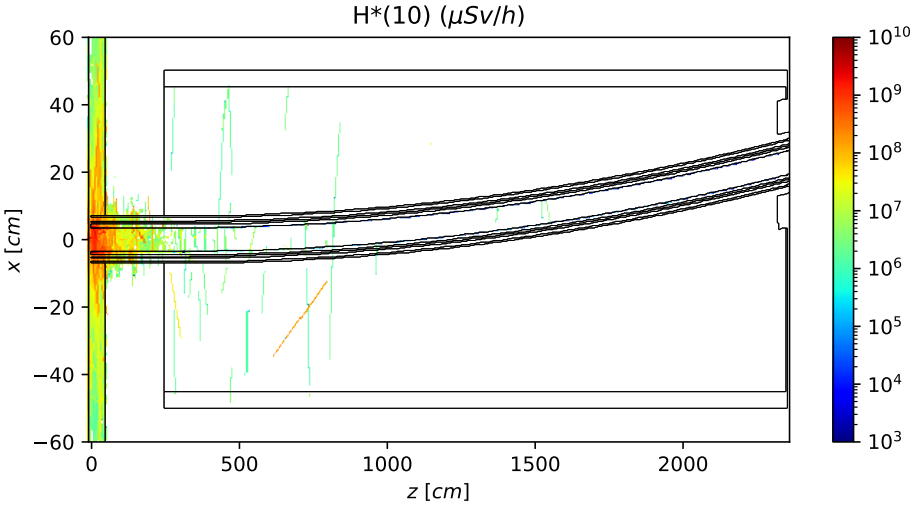
Las observaciones coinciden con las realizadas anteriormente, y nuevamente corroboran lo esperado teóricamente según el estudio del Apéndice B. En la Figura 4.26 se comparan las curvas de dosis sobre un corte longitudinal, sobre $x = -10\ cm$, donde se verifica que los niveles de dosis con la fuente KDE coincide con el de la fuente de *tracks* con repetición. Finalmente, en la Figura 4.27 se comparan las curvas longitudinales de dosis, sobre $x = -10\ cm$, para la fuente a la entrada de la guía y la fuente sobre los espejos, proveniente del acople óptica-neutrónica. Se observa que la dosis por neutrones coincide, lo cual es esperable considerando que los neutrones rápidos, principales responsables de esta, no interactúan con los espejos. Por otra parte, para la dosis por fotones se observa un notable desplazamiento de la curva hacia valores con mayor z, lo cual se debe a que los neutrones térmicos, principales causantes de los fotones *prompt*, avanzan una mayor distancia antes de escapar de la guía y ser absorbidos, al realizar el acople con McStas. Ambos gráficos están promediados en un intervalo de 25 mm alrededor de $x = -10\ cm$.

En esta simulación también se registraron dos *tallies* de activación en Al-27 (sistema de alineación) y Fe-58 (sistema de vacío), los cuales se muestran en la Figura 4.28. Se consideraron únicamente estos dos isótopos pues suelen ser los que aportan la mayor contribución a la radiación por decaimientos. Posteriormente, los *tallies* volumétricos de activación fueron convertidos a fuentes KDE volumétricas, con las cuales se realizó una nueva simulación, registrando la dosis por fotones de activación. El mapa de dosis obtenido se muestra en la Figura 4.29, para 10^9 fotones producidos.

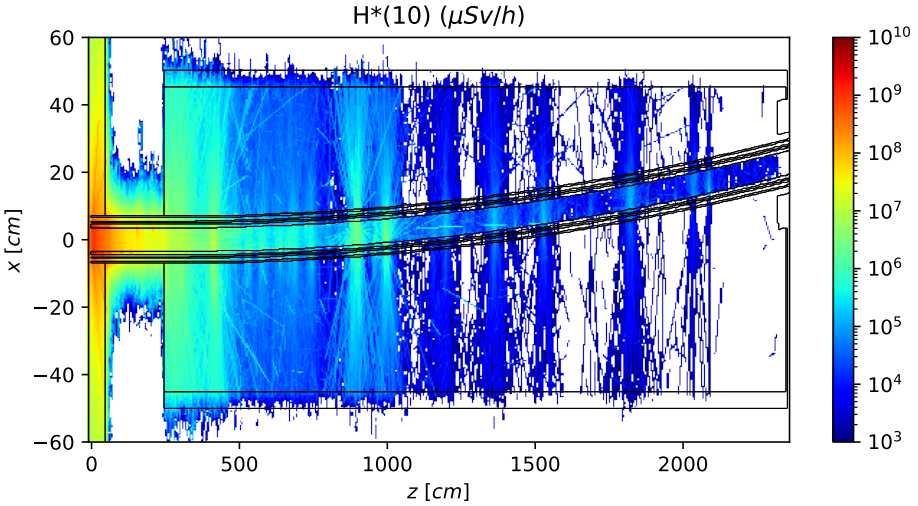
Durante el ajuste de las fuentes KDE de activación, de modo similar a lo realizado con las fuentes de fotones emitidos por absorciones en el sustrato de las guías, se emplearon factores de importancia de 10^{-3} (casi nulos), para la energía y la dirección, ya que luego de la optimización se fijó en cero el *scaling* en energía, y en infinito los de dirección. De este modo se logra representar exactamente el espectro de emisión del Al-28 y Fe-59, y se tiene una emisión isotrópica.

Finalmente, se realizó una corrida empleando la fuente de fotones de fuente, provenientes del tanque de reflector, registrando en este caso únicamente dosis ambiental. El mapa de dosis obtenido se muestra en la Figura 4.30, para 10^9 partículas producidas.

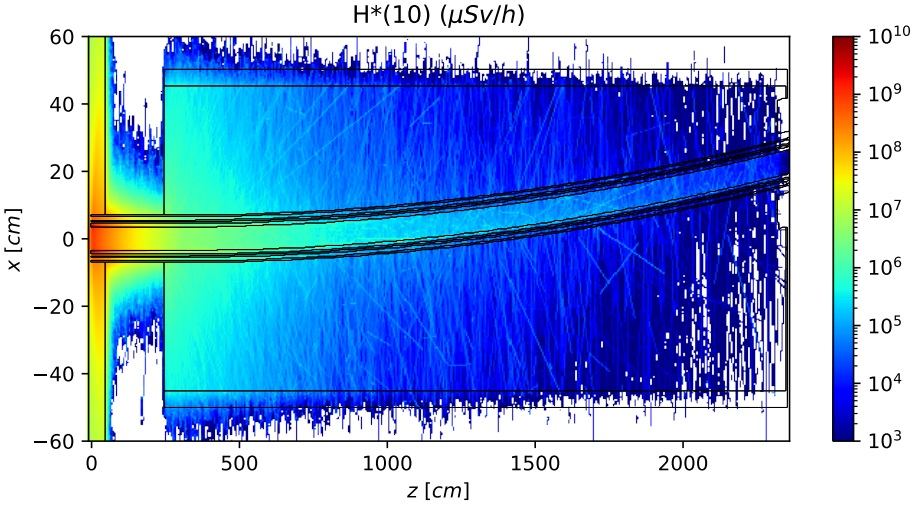
Con los fotones de fuente se completan las cuatro componentes de dosis consideradas, habiéndose mostrado sus mapas de dosis en las Figuras 4.24, 4.25, 4.29 y 4.30. Se puede observar, como era de esperarse, que la zona más comprometida del búnker es la región más cercana al reactor. Para observarla en detalle, en la Figura 4.31 se muestran las dosis por las mismas cuatro partículas, pero en este caso en una sección transversal de la guía, promediada entre los planos $z = 350$ cm y $z = 450$ cm. Además, en la Figura 4.32 se comparan las cuatro componentes sobre un corte transversal en $-5\ cm < y < 5\ cm$. Se observa que, en esta región, los aportes de neutrones, fotones *prompt* y fotones de fuente son similares, y casi cuatro órdenes de magnitud superiores



(a) Fuente de *tracks* sin repetición.

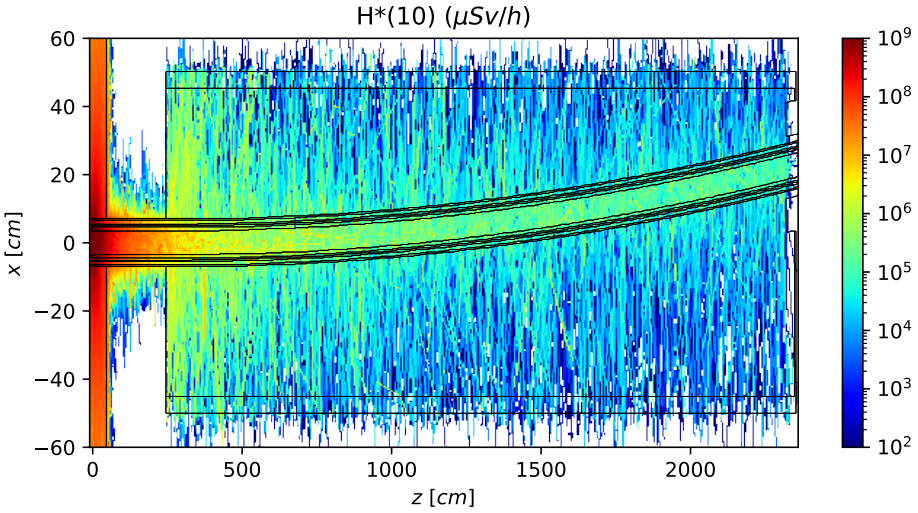


(b) Fuente de *tracks* con repetición.

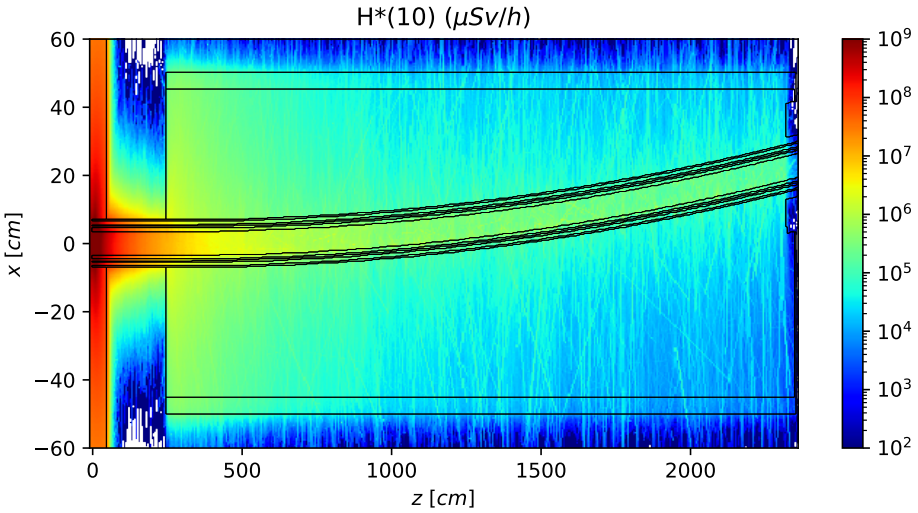


(c) Fuente KDE.

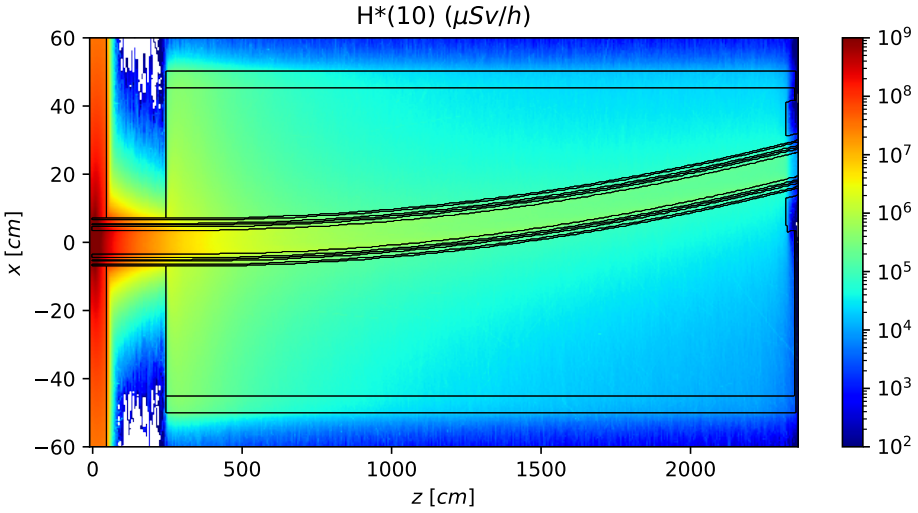
Figura 4.24: Mapas de los dosis por neutrones, con fuentes sobre los espejos de la guía.



(a) Fuente de *tracks* sin repetición.

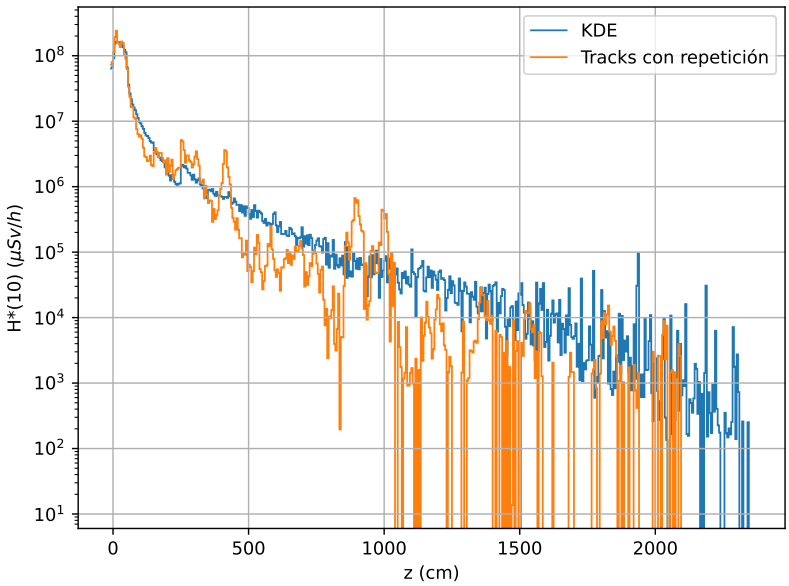


(b) Fuente de *tracks* con repetición.

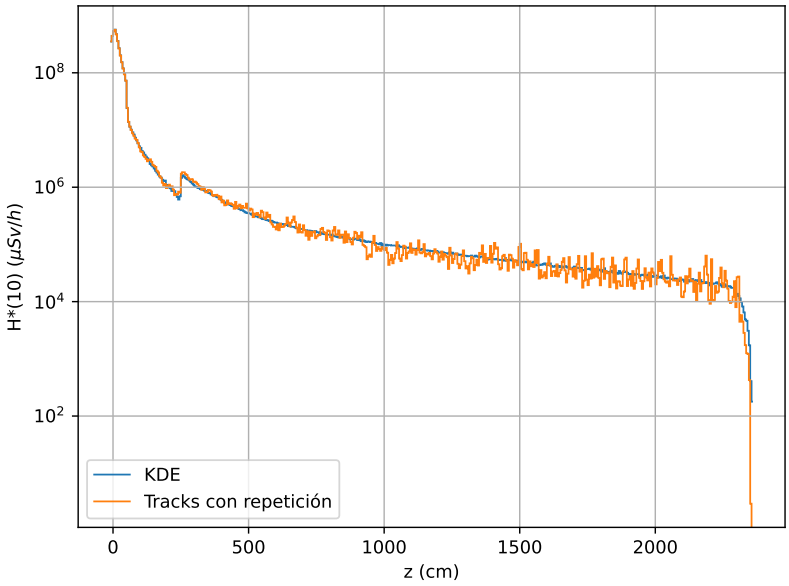


(c) Fuente KDE.

Figura 4.25: Mapas de los dosis por fotones *prompt*, con fuentes sobre los espejos de la guía.

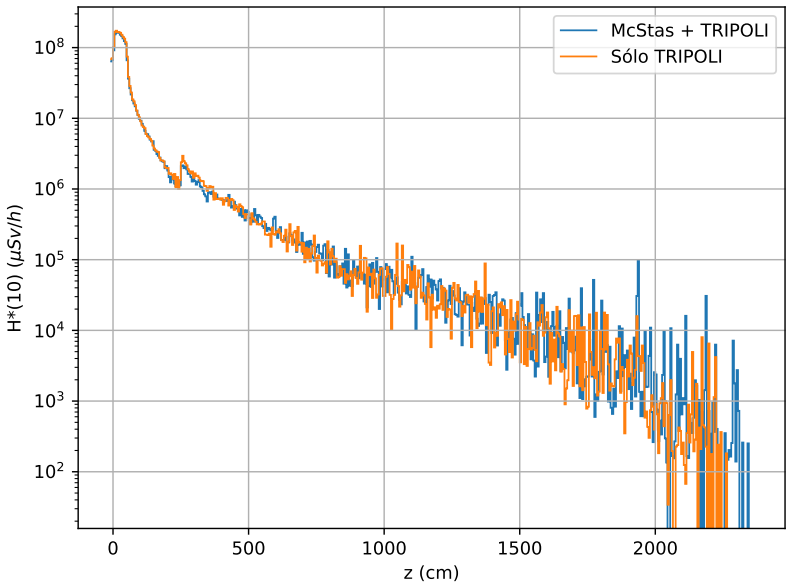


(a) Dosis por neutrones.

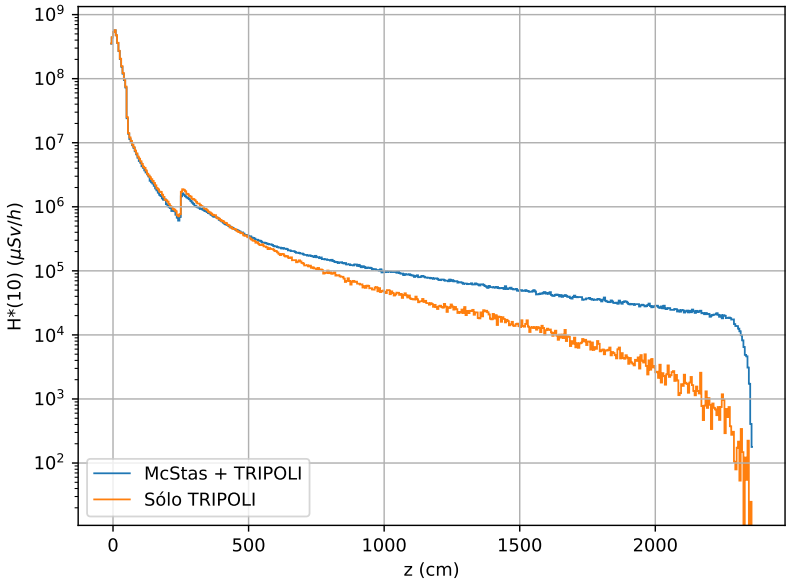


(b) Dosis por fotones *prompt*.

Figura 4.26: Dosis en función de z, sobre $x = -10$ cm, para fuentes de *tracks* con repetición y KDE, sobre los espejos de la guía.

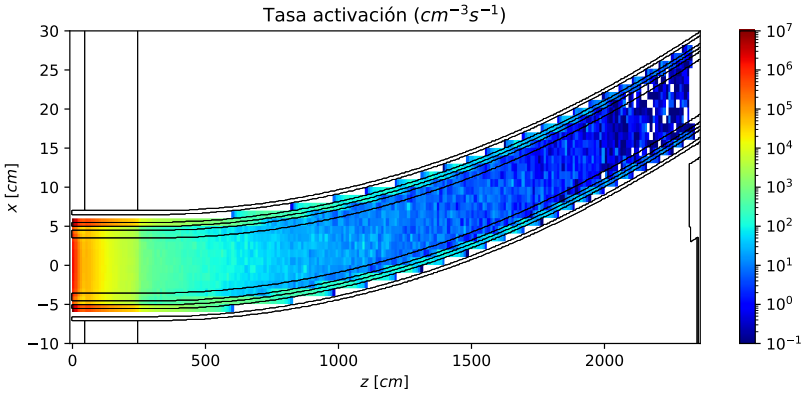


(a) Dosis por neutrones.

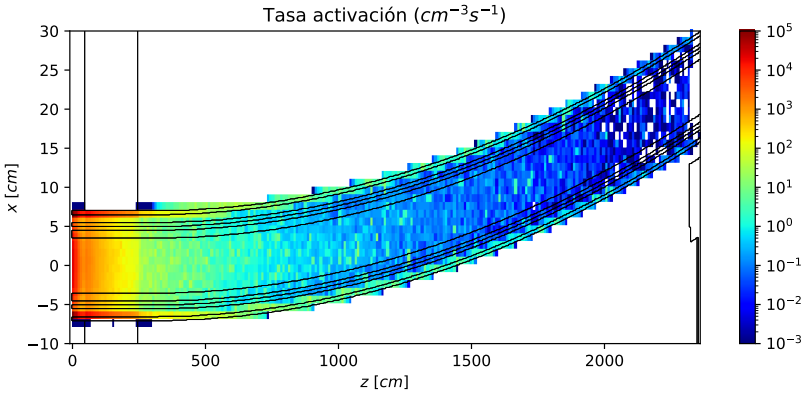


(b) Dosis por fotones *prompt*.

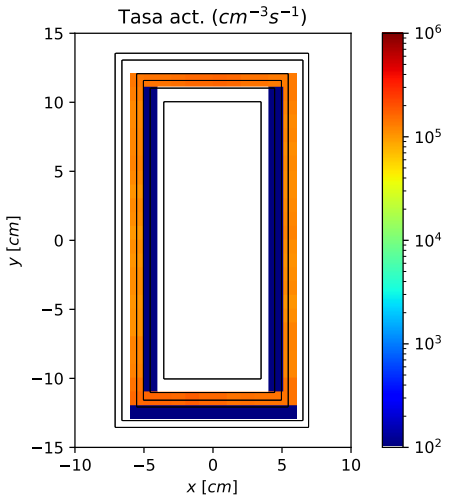
Figura 4.27: Dosis en función de z, sobre $x = -10\ cm$, para fuentes KDE a la entrada de la guía y sobre sus espejos.



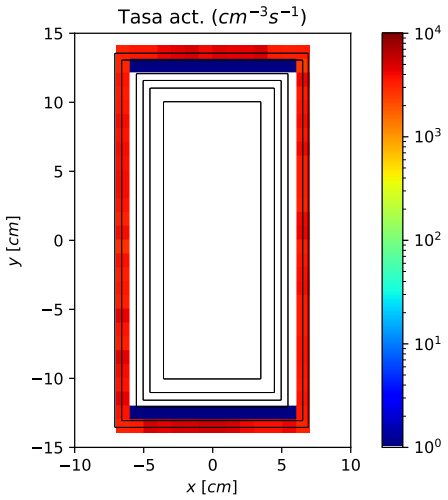
(a) Activaciones de Al-27 promediadas en la dirección y.



(b) Activaciones de Fe-58 promediadas en la dirección y.



(c) Activaciones de Al-27 en un corte transversal, en $z = 150\ cm$.



(d) Activaciones de Fe-58 en un corte transversal, en $z = 150\ cm$.

Figura 4.28: Mapas de los *tallies* de activación registrados, ubicados principalmente en el sistema de alineación (aluminio) y el sistema de vacío (hierro).

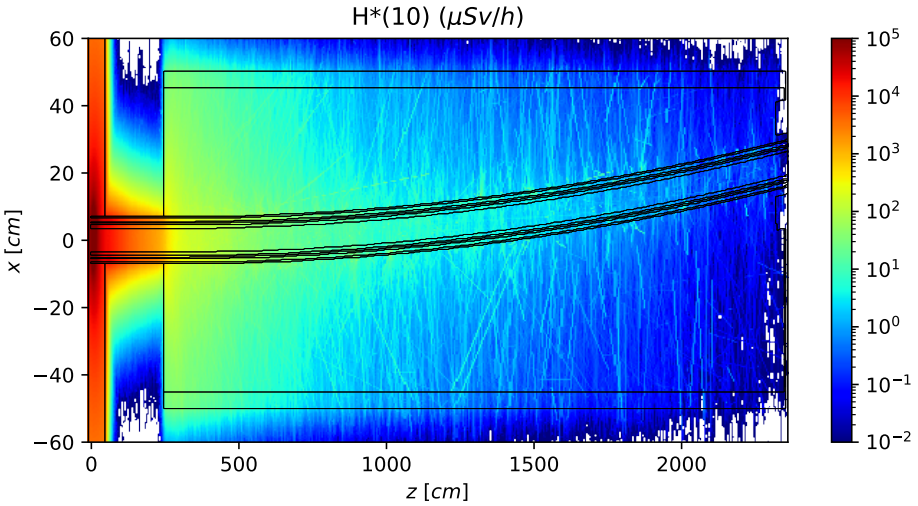


Figura 4.29: Dosis por fotones de activación en un corte horizontal del búnker de guías.

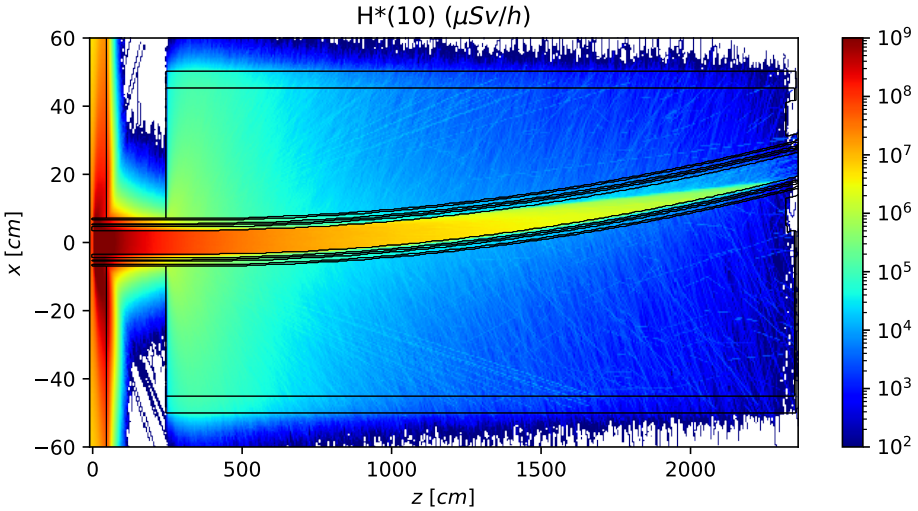
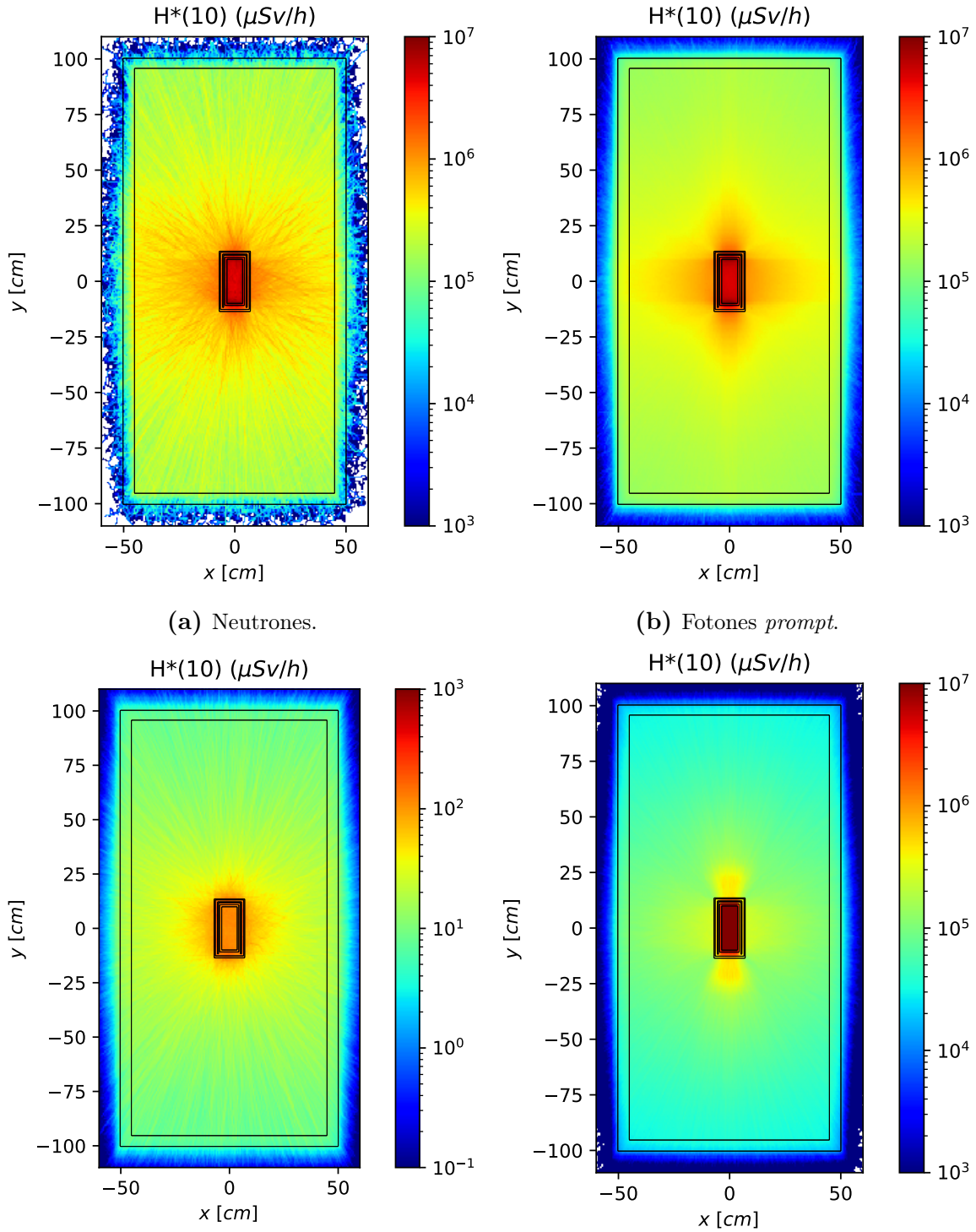


Figura 4.30: Dosis por fotones de fuente en un corte horizontal del búnker de guías.



(a) Neutrones.

(b) Fotones *prompt*.

(c) Fotones de activación.

(d) Fotones de fuente.

Figura 4.31: Dosis ambiental en un corte transversal del búnker de guías, promediada entre en z entre los valores de 350 cm y 450 cm.

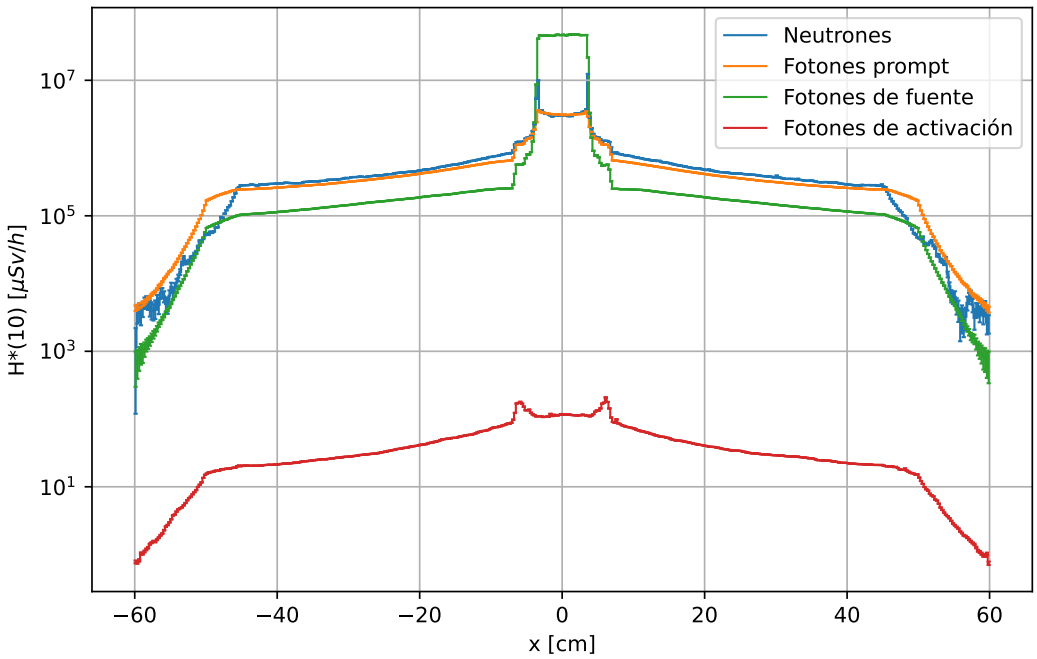


Figura 4.32: Comparación entre las 4 componentes de dosis, promediada en z entre 350 cm y 450 cm, y en y entre -5 cm y 5 cm.

a la dosis por fotones de activación. Cabe mencionar que alrededor de $x = 0$, en el interior de la guía, la dosis por neutrones resulta artificialmente baja, debido a que en TRIPOLI no se simularon los neutrones que viajan por la guía antes de escapar.

4.5.3. Resultados finales

En la Tabla 4.5 se muestra un resumen de las simulaciones realizadas en la cadena de cálculo, así como los recursos computacionales involucrados en cada etapa. En todos los cálculos se utilizaron procesadores Intel Core i7. Por último, en las Figuras 4.33 y 4.34 se muestran los mapas de dosis total, sobre un plano horizontal y uno transversal, en el búnker de guías.

Zona	Part.	Fuente	Programa	Proc.	Part. Gen.	T. cómp. [hs]
Guía	n	KDE	McStas	1	10^8	0,13
Búnker	n, γ_{prompt}	*Tracks*	TRIPOLI	1	10^9	16,75
Búnker	n, γ_{prompt}	KDE	TRIPOLI	1	10^9	24,13
Búnker	γ_{fuente}	KDE	TRIPOLI	8	10^9	27,91
Búnker	γ_{activ}	KDE	TRIPOLI	8	10^9	1,90
Total						70,82

Tabla 4.5: Esquema de las simulaciones realizadas. Para cada simulación se muestra la zona, partículas, tipo de fuente, programa, cantidad de procesadores utilizados, cantidad de partículas generadas y tiempo de cálculo.

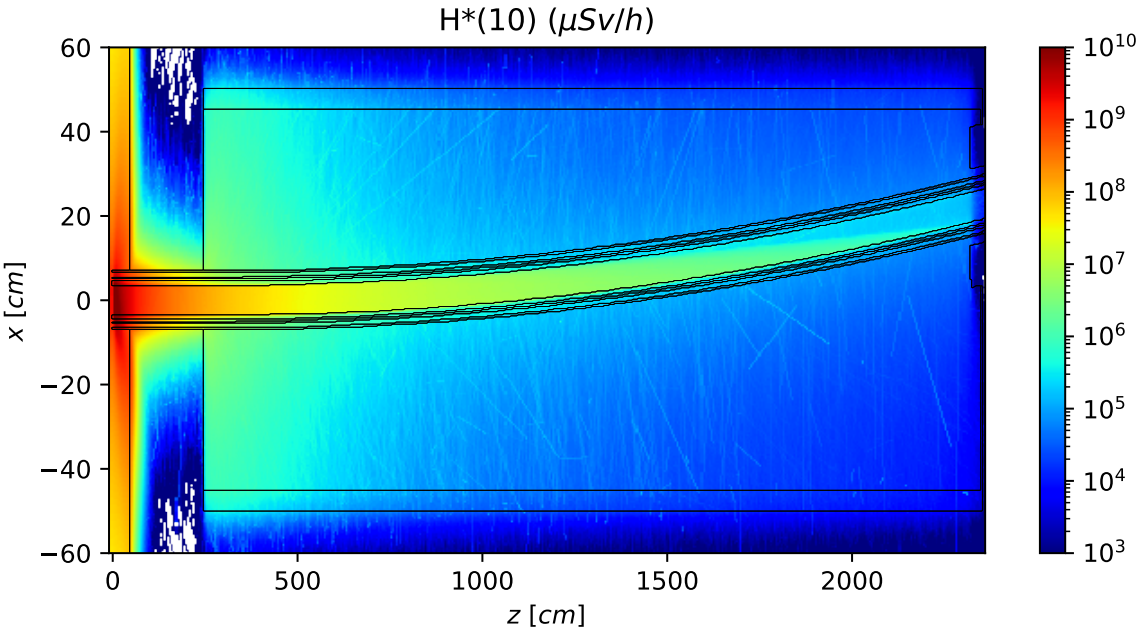


Figura 4.33: Dosis total en un plano horizontal del búnker, promediada entre y = -5 cm e y = 5 cm.

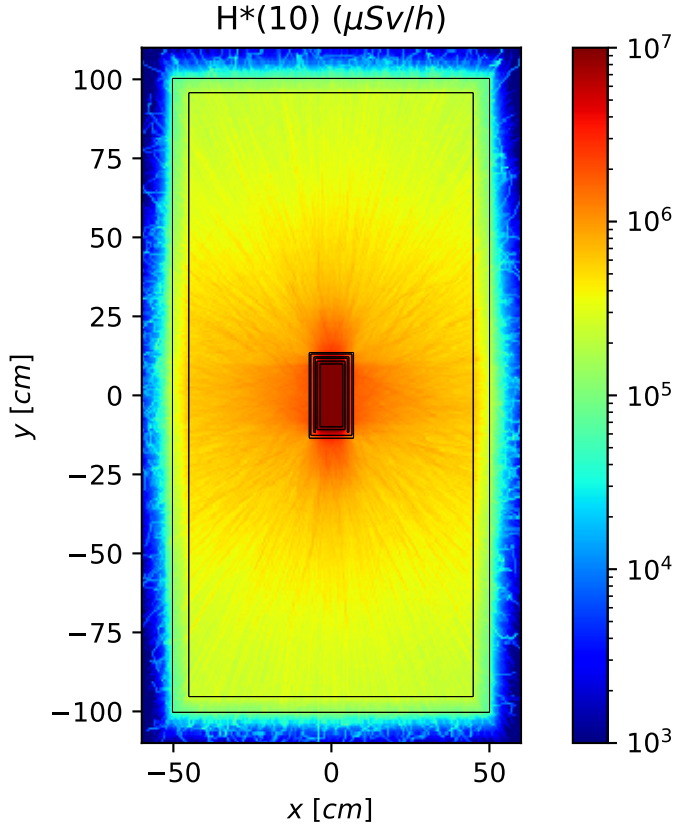


Figura 4.34: Dosis total en un plano transversal del búnker, promediada entre z = 350 cm y z = 450 cm.

Capítulo 5

Conclusiones

En el presente trabajo se construyó una herramienta computacional, denominada KDSource, para el modelado y uso de fuentes de distribuciones en códigos Monte Carlo, por el método *Kernel Density Estimation*. Los desarrollos continúan el camino de lo realizado en mi Proyecto Integrador, que a su vez continúa los de Fairhurst y Ayala, pero provee numerosas ventajas gracias al método KDE, en comparación con los histogramas utilizados anteriormente. La herramienta implementada permite la creación, optimización y caracterización de fuentes distribucionales gracias a su API en Python, y el guardado del modelo resultante en un archivo XML. Este luego se utiliza para la generación de nuevas partículas respetando la distribución estimada, sin límite en la cantidad de muestreos, mediante la API en C o bien a través de la herramienta de línea de comando.

Mediante la herramienta KDSource es posible modelar diferentes geometrías de fuente, y elegir los tratamientos deseados para la energía y el ángulo. Con respecto a la geometría, es posible modelar fuentes planas, especialmente adecuadas para haces de partículas pero también para interfaces en cálculos de blindajes, fuentes volumétricas, orientadas a las fuentes de activación, y fuentes tipo guías, las cuales registran los escapes a través de los espejos de una guía neutrónica. Con respecto a la energía, es posible elegir entre un tratamiento en letargía, útil para neutrones, o en energía, más adecuado para fotones. Finalmente, para la distribución angular es posible una parametrización con variables polares, apropiada para fuentes planas, o bien un tratamiento isotrópico, ideal para fuentes de emisión por decaimiento post-activación.

El paquete KDSource se encuentra integrado con el formato de listas de partículas MCPL. Gracias a este, y a las modificaciones realizadas mediante desarrollo propio, se logra la posibilidad de comunicación con una amplia variedad de códigos Monte Carlo comerciales, además de aprovecharse las variadas funcionalidades y bondades de dicho formato. Por otra parte, también se empleó la biblioteca de Python `KDEpy` para el método KDE, complementada con algunas funcionalidades añadidas externamente.

El objetivo de este trabajo es facilitar los cálculos de blindajes, por lo que se incluyó en KDSource una serie de archivos plantilla que asisten las diferentes tareas requeridas. Se dio particular atención al acople entre las dos principales códigos Monte Carlo utilizados: McStas y TRIPOLI. Gracias a los diferentes recursos de asistencia y acople, complementado por la documentación provista en el manual de `KDSource`, incluido en la distribución, se logra una reducción en las horas de ingeniería requeridas para la utilización de fuentes de distribuciones en un cálculo de blindajes o de haces de partículas.

La utilidad de la herramienta se demostró en un cálculo conceptual de blindajes en la periferia de una guía neutrónica, basado en el haz GF1 de RA10. Se utilizaron los mismos modelos empleados en mi Proyecto Integrador, y se verificó la consistencia de los resultados. El modelo incluyó el acople de óptica neutrónica simulada en McStas con el transporte de radiación modelado en TRIPOLI. Además, se observó la reducción de varianza que aporta la herramienta, permitiendo la obtención de mapas de dosis en regiones lejanas al núcleo.

Por último este trabajo también permitió la consolidación de la línea de investigación asociada a fuentes distribucionales, en un paquete de *software* libre disponible para el uso y colaboración en la plataforma GitHub. De este modo se provee unificación a los aportes realizados en distintos trabajos de grado y posgrado en el Instituto Balseiro, y a los que se espera que sigan realizándose en el futuro. La consolidación también implica un nuevo nivel de alcance de los desarrollos a posibles usuarios y colaboradores externos, gracias al empleo de estándares de producción y distribución del código, como `git`, `pip` y `cmake`.

Existen numerosas posibles mejoras al paquete `KDSource`, algunas de las cuales se presentan a continuación. Con respecto a la metodología de los cálculos de blindajes, resultaría interesante un análisis mayor de los diferentes errores asociados a las incertezas de la distribución de fuente, y su observación en simulaciones específicas. También es posible un mayor estudio del efecto que causan las absorciones neutrónicas en los espejos de las guías, especialmente en el níquel, lo cual fue modelado en este trabajo pero no analizado en detalle. Con respecto a la arquitectura computacional, podrían resultar beneficias ciertas modificaciones a la biblioteca `KDEpy`, lo cual podría resultar en un librería independiente basada en una bifurcación (*fork*). Por otra parte, también es posible aplicar los conceptos de estimación de densidad a problemas con dependencia temporal, lo cual abre un nuevo campo de aplicaciones, como el modelado de *choppers*. Por último, sería útil la contar con la posibilidad de emplear de otros *kernels*, distintos al *gaussiano*, en el método KDE, especialmente el de Epanechnikov (parabólico) que al ser finito podría representar más adecuadamente los bordes de las fuentes, sin desmedro en la capacidad de suavizado.

Apéndice A

Documentación de la biblioteca KDSource

A.1. Descripción del formato de archivo de parámetros XML

Un archivo de parámetros XML es el método por el cual se puede almacenar una fuente KDSource en el disco. Usualmente dicho archivo es generado mediante la API en Python, luego de la optimización del ancho de banda. Este formato sirve además como método de comunicación entre los distintos componentes de KDSource, ya que puede utilizarse para reconstruir una fuente tanto en Python como en C, y para aplicar re-muestreo por línea de comando.

Como se muestra en el ejemplo del Listing A.1, un archivo de parámetros posee la siguiente información:

- Corriente total de la fuente, en unidades $[1/s]$.
- `PList`: Lista de partículas.
  - Tipo de partícula global.
  - Archivo MCPL con la lista de partículas.
  - Traslación (opcional).
  - Rotación (opcional).
  - Transformación $(x, y, z) \rightarrow (y, z, x)$ (opcional).
- `Geom`: Geometría y tratamiento de variables. El orden indica la cantidad de submétricas.
  - Submétricas. Para cada una, además de su nombre, se indica:

        - Dimensionalidad.
        - Parámetros (específicos para cada métrica).
    - Traslación (opcional).
    - Rotación (opcional).

- Factores de escaleo para cada variable de parametrización.
- Ancho de banda. Si es constante (`variable="0"`) se indica su valor, mientras que si es adaptativo (`variable="1"`) se indica el *path* del archivo conteniendo la lista de valores. Dicho archivo debe poseer una secuencia de valores en formato binario de punto flotante de simple precisión (32 bits), sin separación, de la misma longitud que la lista de partículas en el archivo MCPL.

Listing A.1: Ejemplo de archivo de parámetros XML.

```
<?xml version="1.0" ?>
<KDSource>
    <J units="1/s">1.0</J>
    <PList>
        <pt>n</pt>
        <mcplname>/path/to/mcplfile.mcpl.gz</mcplname>
        <trasl>  0.   0. -10.</trasl>
        <rot/>
        <x2z>0</x2z>
    </PList>
    <Geom order="3">
        <Lethargy>
            <dim>1</dim>
            <params nps="1">10.0</params>
        </Lethargy>
        <SurfXY>
            <dim>2</dim>
            <params nps="5">-50. 50. -50. 50. 0.</params>
        </SurfXY>
        <PolarMu>
            <dim>2</dim>
            <params nps="0"/>
        </PolarMu>
        <trasl>10.  0. 50.</trasl>
        <rot>0.         3.14159265 0.        </rot>
    </Geom>
    <scaling>2.2339 14.1445 10.0147 0.2362 103.6779</scaling>
```

```
    <BW variable="1">/path/to/bwfile</BW>
</KDSource>
```

A.2. Documentación de aplicación kdtool

La aplicación de linea de comando de KDSource se accede a través del comando `kdtool`. Sus instrucciones de uso se pueden obtener mediante el argumento `--help`, y se muestran en el Listing A.2.

Listing A.2: Instrucciones de uso del comando `kdtool` (output de "`kdtool -help`").

```
Usage: kdtool [options]

KDSource is a Monte Carlo calculations assistance tool. It implements particles
density estimation and sampling by means of Kernel Density Estimation method.

Options:
    resample:   Resample particles based on a kdsource XML file.
    templates:  Copy templates for Monte Carlo calculations.
    beamtest:   Test source with simple beam calculation.
    [Any MCPL command]
    -h, --help: Display usage instructions.
```

La opción `resample` de `kdtool` permite generar nuevas muestras en base a un modelo KDSource guardado en un archivo de parámetros XML. Sus instrucciones de uso se pueden obtener mediante el argumento `--help`, y se muestran en el Listing A.3.

Listing A.3: Instrucciones de uso del comando `kdtool resample` (output de "`kdtool resample -help`").

```
Usage: kdtool resample sourcefile [options]

Resample particles from source defined in XML file sourcefile, and save them in
a MCPL file.

Options:
    -o outfile: Name of MCPL file with new samples
                (default: \"resampled.mcpl\").
    -n N:       Number of new samples (default: 1E5).
    -h, --help: Display usage instructions.
```

La opción `templates` de `kdtool` permite copiar los archivos plantilla (Jupyter Notebooks) de las operaciones más usuales con la API de Python al directorio de trabajo. Además, opcionalmente, permite copiar plantillas para ejecutar McStas o TRIPOLI-

4 en acople con KDSource. Sus instrucciones de uso se pueden obtener mediante el argumento `--help`, y se muestran en el Listing A.4.

Listing A.4: Instrucciones de uso del comando `kdtool templates` (output de "`kdtool templates -help`").

```
Usage: kdtool templates dest [options]

Copy to dest templates for KDSource usage in Python, or for interacting with
Monte Carlo codes.

Options:
    --mcstas:   Copy templates for using McStas.
    --tripoli:  Copy templates for using TRIPOLI-4.
    --all:      Copy all templates.
    -h, --help: Display usage instructions.
```

La opción `beamtest` permite evaluar la validez de la fuente KDE en un cálculo de haces, comparando su funcionamiento con la fuente de *tracks* (muestreo directo de partículas de la lista). Su uso está orientado a fuentes planas de neutrones, y la comparación consiste en una simulación en la cual se propagan las partículas de fuente hasta un colimador rectangular, donde se mide la corriente. Dicha simulación se ejecuta dos veces, una con la fuente KDE y una muestreando las partículas directamente de la lista, y los resultados se almacenan en un documento de texto. Es posible configurar la posición y tamaño del colimador rectangular, para evaluar la concordancia entre ambas fuentes en distintos casos. Sus instrucciones de uso se pueden obtener mediante el argumento `--help`, y se muestran en el Listing A.5.

Listing A.5: Instrucciones de uso del comando `kdtool beamtest` (output de "`kdtool beamtest -help`").

```
Usage: kdtool beamtest sourcefile [options]

Executes a simple simulation with source defined in XML file sourcefile, in
which calculates the number of particles passing thru a rectangular collimator.
The simulation is repeated using the particle list directly as source, to
compare the results.

Results are computed for 4 energy groups, and stored in a results file which
can be imported from a spreadsheet.

This tool is designed to be used with flat neutron sources with particles
propagating towards z direction.

Options:
```

```
-n N:           Number of source particles (default: 1E6).
-o results:     Name of file to store results.
-xwidth value:  Width of the collimator, in cm (default: 7).
-yheight value: Height of the collimator, in cm (default: 20).
-z value:       Position of the collimator along z axis, in cm
                (default: 500).
-xshift:        Horizontal shift of the center of the collimator,
                in cm (default: 0)
-yshift:        Vertical shift of the center of the collimator,
                in cm (default: 0)
-h, --help:     Display usage instructions.
```

Por último, el comando `kdtool` permite acceder a cualquiera de las aplicaciones de línea de comando de MCPL, en su versión extendida incluida en el paquete KDSource.

A.3. Documentación de API en Python

La API en Python de KDSource, denominada `kdsource`, se compone de los siguientes módulos:

- `kdsource.py`: Módulo para el objeto `KDSource`, que representa una fuente KDE.
- `kde.py`: Módulo para métodos de selección de ancho de banda, para la biblioteca `KDEpy`.
- `plist.py`: Módulo para el objeto `PList`, y operaciones sobre listas de partículas.
- `geom.py`: Módulo para el objeto `Geometry`, la clase abstracta `Metric`, y todas sus implementaciones heredadas.
- `stats.py`: Módulo para análisis estadístico de listas de partículas, mediante el objeto `Stats`.
- `summary.py`: Módulo para colectar los principales resultados de simulaciones Monte Carlo, mediante el objeto `Summary`.
- `tally.py`: Módulo para lectura, gráficos, y conversión a lista de partículas, de *tallies* de TRIPOLI-4, mediante el objeto `T4Tally`.
- `utils.py`: Utilidades generales.

Todas las funcionalidades de los módulos mencionados se encuentran presentes en el *namespace* principal de la biblioteca `kdsource`.

A.3.1. Ejemplo de uso

Listing A.6: Ejemplo básico de uso de API en Python.

```
import kdsource as kds

# Define particle list
plist = kds.PList("surfsource", readformat="ssw")
# Define geometry
geom = kds.GeomFlat()
# Create KDSource
s = kds.KDSource(plist, geom, bw="mlcv")

# Fit KDSource
s.fit(N=1E5)

# Save in XML file
s.save("source.xml")
```

En el Listing A.6 se muestra un ejemplo básico de uso. Las principales etapas son:

- Creación de objeto `KDSource` en base a objetos `PList` y `Geometry`.
- Ajuste de anchos de banda.
- Gráficos de distribuciones.
- Guardado de la fuente en archivo XML.

En los archivos plantilla (disponibles mediante "`kdtool templates .`") se muestran más ejemplos de uso de los distintos módulos de KDSource.

A.3.2. Módulo `kdsource`

Este módulo se centra en la clase `KDSource`, que modela una fuente de partículas KDE. Para su creación se debe contar previamente con objetos `PList` y `Geometry`. Mediante un modelo `KDSource` se puede aplicar el método KDE sobre una lista de partículas, optimizar el ancho de banda, generar gráficos de las distribuciones estimadas, y guardar el modelo en un archivo XML, para su posterior uso con la misma u otras APIs.

El objeto `KDSource` sirve como *wrapper* de un modelo KDE de la biblioteca `KDEpy`, el cual guarda en el parámetro `kde`. Es posible modificar manualmente el ancho de banda modificando el parámetro `bw` de dicho objeto. Además `KDSource` posee un parámetro `scaling`, el cual contiene los factores de normalización para cada variable (ver 3.3.4).

En el Listing A.6 se muestra un ejemplo de uso básico del objeto KDSource, mientras que en el Listing A.7 se muestra el uso de las funciones de gráficos. Es posible realizar gráficos tanto 1D como 2D, y para cada uno de ellos existen 2 posibilidades:

- Gráficos integrados: Se grafica la densidad en función de 1 ó 2 variables parametrizadas, integrando sobre las demás (en un rango finito o en todo su dominio). Este es el caso de `plot_integr`, `plot_E` y `plot2D_integr`. Es el método de graficación recomendado. La función `plot_E` grafica el espectro en función de la energía independientemente de la parametrización elegida.
- Gráficos puntuales: Se grafica la densidad conjunta en función de 1 ó 2 variables no parametrizadas. Este es el caso de `plot_point` y `plot2D_point`.

Listing A.7: Ejemplo de gráficos de distribuciones estimadas.

```
# Energy plot
EE = np.logspace(-9,1,50)
fig,[scores,errs] = s.plot_E(EE)
plt.show()

# Theta plot
tt = np.linspace(0,180,50)
fig,[scores,errs] = s.plot_integr('theta', tt)
plt.show()

# XY plot of epithermal neutrons
umin = 3  # Minimum lethargy
umax = 16 # Maximum lethargy
# Vector of min and max vals (parametrized)
#      [u   , x     , y     , theta, phi]
vec0 = [umin,-np.inf,-np.inf, 0    ,-180]
vec1 = [umax, np.inf, np.inf, 180  , 180]
xx = np.linspace(-10,10,30)
yy = np.linspace(-10,10,30)
s.plot2D_integr(['x','y'], [xx,yy], vec0=vec0, vec1=vec1)
plt.show()

# XY plot at fixed energy and angle
# Vector of fixed values (non parametrized)
#       [E   , x, y, z, dx, dy, dz]
part0 = [1E-3, 0, 0, 0, 0 , 0 , 1 ]
xx = np.linspace(-10,10,30)
yy = np.linspace(-10,10,30)
```

```
s.plot2D_point(['x','y'], [xx,yy], part0=part0)
plt.show()
```

A.3.3. Módulo `kde`

Este módulo contiene métodos de selección de ancho de banda que complementan la biblioteca `KDEpy`. Debido a que esta no admite anchos de banda multidimensionales, y gracias a la normalización de datos que aplica la clase `KDSource`, los métodos en este módulo asumen que las dispersiones de los datos en cada variable son unitarios, y obtienen anchos de banda unidimensionales. Si se admite, desde luego, ancho de banda adaptativo, es decir uno por cada partícula. Se incluyen 3 técnicas de optimización:

- Regla de Silverman: Se obtiene el ancho de banda en función de N_{eff} y dim mediante el método `bw_silv`.

- K Vecinos Más Cercanos (KNN): Se obtiene un ancho de banda adaptativo como la distancia al K-ésimo vecino de cada partícula, mediante el método `bw_knn`. El cálculo se realiza por *batches*, y se debe especificar o bien la cantidad de vecinos por *batch* o bien la cantidad estimada total. Si la cantidad resultante de vecinos por *batch* no es entera se utiliza un factor de ajuste f.

- Validación Cruzada de Máxima Probabilidad (MLCV): Se evalúa el *score* de log-probabilidad media con un esquema de validación cruzada, para cada valor en una grilla de anchos de banda. Se toma el ancho de banda que maximiza dicho *score*. Es posible especificar tanto el ancho de banda semilla como la grilla de factores a utilizar para construir la grilla de anchos de banda. Se obtiene el ancho de banda óptimo con el método `bw_mlcv`.

El método recomendado es la MLCV, utilizando una semilla proveniente de KNN, aunque también es el más costoso. De todos modos, el método más adecuado puede variar según el problema. Nótese que, si se pretende guardar la fuente KDE y utilizarla para muestrear nuevas partículas, el método KNN, y el MLCV que lo utilice como semilla, debe realizarse utilizando todas las partículas en la lista (argumento "`N=-1`" en función `fit`), pues de lo contrario habría más partículas que anchos de banda para el KDE adaptativo.

Por último, el método `optimize_bw` sirve como *wrapper* de los distintos métodos de optimización, redirigiendo a a cada uno correspondientemente. Este método es llamado por la clase `KDSource` al momento del ajuste.

A.3.4. Módulo `plist`

Este módulo se centra en la clase `PList`, la cual sirve como *wrapper* de listas de partículas en formato MCPL. Su principal función es permitir acceder a las partículas almacenadas, convirtiéndolas a formato de `numpy`. Además incluye la posibilidad de aplicar una traslación y una rotación a las partículas apenas luego de leerlas, lo cual es útil al cambiar de sistema de referencia entre simulaciones. En el Listing A.8 se muestra un ejemplo de uso.

Es posible emplear de manera conjunta más de una lista de partículas, siempre y cuando tengan todas el mismo formato. Los formatos posibles son `"mcpl"`, `"ssw"` (MCNP), `"phits"`, `"ptrac"` (MCPN), `"stock"` (TRIPOLI-4) y `"ssv"` (ASCII).

Listing A.8: Ejemplo de uso de `PList`.

```
file1 = "surfsource1"
file2 = "surfsource2"
trasl = [0, 0, -20]
rot = [0, np.pi/2, 0]

# Create PList
pl = kds.PList([file1,file2], readformat='ssw', pt='n', trasl=trasl, rot=rot)

# Get particles and weights
parts,ws = pl.get(1E4)
```

Otras funcionalidad presentes en el módulo son:

- `convert2mcpl` y `join2mcpl`: Permiten convertir listas de partículas de cualquier formato compatible con MCPL a MCPL. Internamente ejecutan los comandos de conversión de línea de comando.
- `savessv` y `appendssv`: Permiten guardar *arrays* de partículas en formato de `numpy` en archivos ASCII SSV. Estos pueden luego ser convertidos a MCPL, por ejemplo mediante `convert2mcpl`.

A.3.5. Módulo `geom`

Este módulo se centra en las clases `Geometry` y `Metric`. La clase `Geometry` representa un conjunto de tratamientos a las variables que definen una partícula. La misma se compone de un conjunto de métricas, además de una posición y rotación que definen la ubicación espacial de la fuente. Las métricas definen el tratamiento de cada conjunto de variables (energía, posición y dirección) a través de una transformación de parametrización.

La clase `Metric` es una clase abstracta, con las siguientes herencias implementadas:

- `Energy`: Tratamiento simple para la energía, sin transformación.
- `Lethargy`: Métrica de letargía, definida como $u = log(E_0/E)$.
- `Vol`: Tratamiento simple para posición, para fuentes volumétricas.
- `SurfXY`: Tratamiento simple para posición, para fuentes planas en XY.
- `Guide`: Tratamiento conjunto para posición y dirección, para fuentes con geometría de guía.
- `Isotrop`: Métrica simple para la dirección, basada en la distancia angular entre direcciones.
- `Polar`: Métrica polar para la dirección, con ángulos θ (distancia angular al eje z) y ϕ (ángulo acimutal con respecto al eje x).
- `PolarMu`: Métrica polar, con $\mu = cos(\theta)$.

Además, se definieron las siguientes funciones para la creación rápida de las geometrías más usuales.

- `GeomFlat`: Fuente plana. Métricas: `Lethargy`, `SurfXY` e `Isotrop`.
- `GeomGuide`: Fuente sobre espejos de guía. Métricas: `Lethargy` y `Guide`.
- `GeomActiv`: Fuente volumétrica. Métricas: `Energy`, `Vol` e `Isotrop`.

En el Listing A.9 se muestra un ejemplo de uso de `Geometry`.

Listing A.9: Ejemplo de uso de `Geometry`.

```
# Metrics
m_E = kds.Lethargy(E0=20)
m_pos = kds.SurfXY(z=5)
m_dir = kds.Polar()

# Create Geometry
trasl = [0,0,15]
rot = [0,np.pi,0]
geom = kds.Geometry([m_E,m_pos,m_dir], trasl=trasl, rot=rot)

# Use Geometry
vecs_param = geom.transform(parts)
parts = geom.inverse_transform(vecs_param)
mean = geom.mean(parts=parts, weights=ws)
std = geom.std(parts=parts, weights=ws)
```

A.3.6. Módulo `stats`

Este módulo se centra en la clase `Stats`, la cual permite realizar un análisis de algunos indicadores estadísticos, y su relación con la cantidad de partículas. Sirve como complemento de las herramientas incluidas en la API de MCPL para tal fin, más precisamente las funciones `collect_stats`, `dump_stats` y `plot_stats`.

En el Listing A.10 se muestra un ejemplo de uso de la clase `Stats`. La misma se inicializa con una lista de partículas obtenida de `PList`, y permite graficar la variación de parámetros estadísticos con el número de partículas. A través del argumento `steps` se regula la cantidad de subconjuntos, de tamaño creciente, para los cuales se evaluará el parámetro estadístico correspondiente. Por ejemplo, para `steps=2` la evaluación se realizaría para la mitad de la lista de partículas, y para la lista completa. Los gráficos obtenidos permiten observar el grado de convergencia de los parámetros de interés para el número de partículas en la lista.

Listing A.10: Ejemplo de uso de `Stats`.

```
stats = kds.Stats(parts, ws)

N,I,err = stats.mean_weight(steps=100)
plt.show()
N,mn,err = stats.mean(var=1, steps=100)
plt.show()
N,std,err = stats.std(var=1, steps=100)
plt.show()
```

A.3.7. Módulo `summary`

Este módulo se centra en el objeto `Summary`, y está especialmente pensado para simplificar la transcripción a una planilla de cálculo de los resultados principales en una simulación. Actualmente, los códigos soportados para esta funcionalidad son McStas y TRIPOLI-4.

En el Listing A.11 se muestra un ejemplo de uso de `Summary`. Además, se provee una plantilla para el mismo fin en el archivo `postproc.ipynb`.

Listing A.11: Ejemplo de uso de `Summary`.

```
mccode = "TRIPOLI"            # "McStas" or "TRIPOLI"
folder = "outdir"             # Directory containing simulation outputs
bashoutput = "bash.out"       # Bash output from simulation
n_detectors = [tracks1]       # Neutron list detectors
t4output = "simul.out"        # TRIPOLI output (only if mccode="TRIPOLI")
tallies = ["activ-fe","dose"] # Tally names
```

```
summary = kds.Summary(mccode,
                      folder,
                      bashoutput=bashoutput,
                      n_detectors=n_detectors,
                      t4output=t4output,
                      tallies=tallies)

summary.save("summary.txt")  # Saves inside summary.folder
```

A.3.8. Módulo `tally`

Este módulo está destinado al procesamiento de *tallies* volumétricos. Actualmente sólo está implementado para TRIPOLI-4, a través de la clase `T4Tally`. El mismo tiene dos funciones principales:

- Graficar mapas de dosis.
- Convertir *tallies* de activación a listas de partículas, para generar fuentes de activación.

Para la generación de fuentes de activación se deben proveer además los espectros de decaimiento de los nucleidos activados, los cuales pueden obtenerse del *Live Chart of Nuclides* de IAEA [37].

En el Listing A.12 se muestra un ejemplo de uso de la clase `T4Tally` para un *tally* de activación. El tratamiento de *tallies* de activación, y su uso para generación de fuentes KDE de fotones de decaimiento, se facilita a través del archivo plantilla `preproc_tally.ipynb`. Por otra parte, el uso de `T4Tally` para gráficos de mapas de dosis se muestra en `doseplots.ipynb`.

Listing A.12: Ejemplo de uso de `T4Tally`.

```
t4output = "simul.out" # Simulation output file
tallyname = "activ-fe" # Tally name
spectrum = "decay-fe"  # Energy decay spectrum

tally = kds.T4Tally(t4output, tallyname, spectrum=spectrum)

# Plot tally
[fig, [scores,errs]] = tally.plot2D(['z', 'x'])

# Save as particle list
mcplname = tally.save_tracks("activsource.mcpl")
```

A.3.9. Módulo `utils`

Este módulo posee utilidades auxiliares. Las mismas se describen a continuación:

- Funciones de conversión entre formatos de tipo de partícula (`pt2pdg` y `pdg2pt`): Convierten entre el Código PDG de tipo de partícula [21], y la notación con caracteres (`"n"`: neutrón, `"p"`: fotón, `"e"`: electrón).
- Funciones de *weighting* (`H10`): Sirven para aplicar un peso a las partículas, y así dar más importancia a las que son de más interés para una aplicación en particular. Actualmente sólo está implementado el *weighting* por factor dosimétrico.
- Funciones de *masking* (`Box`): Sirven para restringir la región en el espacio de fases en el que pueden encontrarse las partículas, eliminando las que se encuentren por fuera. Sirve para focalizar el análisis u optimización en una región de interés. Actualmente sólo está implementado el *masking* tipo hipercubo, es decir fijando un rango para cada variable.

A.4. Documentación de API en C

La API en C consiste en la biblioteca compartida `kdsource`, cuya interfaz de usuario está definida en cuatro archivos de cabeceras, los cuales se describen a continuación:

- `kdsource.h`: Se definen las estructuras `KDSource`, que modela una fuente KDE, y `MultiSource`, que modela un conjunto de `KDSource`'s, además de sus funciones de utilización.
- `plist.h`: Se define la estructura `PList`, *wrapper* de listas de partículas MCPL, y sus funciones de utilización.
- `geom.h`: Se definen las estructuras `Geometry` y `Metric`, que definen el tratamiento de variables, y sus funciones de utilización.
- `utils.h`: Utilidades generales.

Además, se incluye el archivo de cabecera `KDSourceConfig.h`, donde se define la versión de `KDSource` mediante las siguientes constantes:

```
#define KDSource_VERSION_MAJOR 1
#define KDSource_VERSION_MINOR 0
#define KDSource_VERSION_PATCH 0
```

En el archivo `kdsource.h` son incluidos los archivos `mcpl.h`, `KDSourceConfig.h`, `plist.h`, `geom.h` y `utils.h`, por lo que sólo es necesario incluir `kdsource.h` para utilizar la biblioteca `kdsource`. Para compilar un programa que utiliza `kdsource` se deben utilizar los siguientes comandos:

```
KDSOURCE=/path/to/kdsourceinstall
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$KDSOURCE/lib
gcc example.c -lkdsource -lmcpl -lm -I$KDSOURCE/include -L$KDSOURCE/lib
```

Donde `/path/to/kdsourceinstall` es el *path* al directorio donde se instaló el paquete `KDSource`.

A.4.1. Estructuras `KDSource` y `MultiSource`

La estructura `KDSource` modela una fuente de partículas KDE. Usualmente se crea en base a un archivo de parámetros XML generado mediante la API en Python, aunque es posible crearla solamente mediante la API en C. La funcionalidad principal de esta estructura es el muestreo de partículas.

La estructura `MultiSource`, por su parte, modela un conjunto de fuentes `KDSource` superpuestas. En cada muestreo se elige aleatoriamente la fuente a emplear, respetando las intensidades relativas de las fuentes. Es posible implementar *source biasing*, fijando las frecuencias de muestreo por separado las intensidades relativas, lo cual implica un ajuste en los pesos de las partículas nacientes.

A continuación se describen las definiciones, estructuras de datos y funciones declaradas en el archivo `kdsource.h`.

```
#define MAX_RESAMPLES 1000
#define NAME_MAX_LEN 256
```

`MAX_RESAMPLES` define el máximo número de intentos al muestrear, es decir que si no se obtiene una partícula válida luego de `MAX_RESAMPLES` intentos se termina la simulación. `NAME_MAX_LEN` define la máxima longitud posible del nombre de un archivo a ser leído con el paquete `KDSource`.

```
typedef double (*WeightFun)(const mcpl_particle_t* part);
```

Definición de función de *weighting*. Puede utilizarse en algunas funcionalidades para aplicar un pesado o *biasing* basado en los parámetros de las partículas.

```
typedef struct KDSource{
    double J;       // Total current [1/s]
    PList* plist;   // Particle list
    Geometry* geom; // Geometry defining variable treatment
} KDSource;
```

Estructura `KDSource`. Modela una fuente KDE, la cual se compone de una lista de partículas (`PList`) y un tratamiento de variables (`Geometry`). Incluye además el valor de corriente total, en unidades $[1/s]$. Dicho valor es utilizado para definir las intensidades relativas en fuentes múltiples `MultiSource`.

```
KDSource* KDS_create(double J, PList* plist, Geometry* geom);
```

Crear estructura `KDSource` en base a estructuras `PList` y `Geometry` previamente creados. Debe definirse también un valor de corriente total (unidades $[1/s]$). Dicho valor es utilizado para definir

las intensidades relativas en fuentes múltiples `MultiSource`, por lo que puede fijarse en 1 si no se planea crear dicha estructura.

```
KDSource* KDS_open(const char* xmlfilename);
```

Cargar fuente `KDSource` en base al archivo de parámetros XML de nombre `xmlfilename`. El mismo usualmente es creado mediante la API en Python.

```
int KDS_sample2(KDSource* kds, mcpl_particle_t* part, int perturb, double
    w_crit, WeightFun bias, int loop);
```

Función principal para muestreo de partículas con una fuente `KDSource`. La partícula muestreada se guarda en `part`. Incluye los siguientes argumentos para configurar el muestreo:

- `perturb`: Si es 0, las partículas se muestrean directo del archivo MCPL sin modificación. Sino, se aplica una perturbación con la distribución del *kernel* y el ancho de banda correspondiente, de acuerdo a la técnica de muestreo con KDE.
- `w_crit`: Si es menor o igual a 0, se fija el peso estadístico de la partícula muestreada como $w = w_0$, siendo w_0 el peso de la partícula original en el archivo MCPL. Si es mayor a 0, se normaliza w a 1, utilizando la siguiente técnica:
  - Si $w_0 < w_{crit}$: Se usa w_0/w_{crit} como probabilidad de tomar la partícula, en lugar de descartarla y avanzar en la lista.
  - Si $w_0 > w_{crit}$: Se usa w_{crit}/w_0 como probabilidad de avanzar en la lista luego del muestreo.

  De este modo, en promedio se utilizará w_0 veces cada partícula en la lista. Se recomienda fijar `w_crit` como el peso medio en la lista.
- `bias`: Función de *weighting* para aplicar *biasing* durante el muestreo. Será ignorada si `w_crit<=0`.
- `loop`: Si es 0, se llama a `exit(EXIT_SUCCESS)` al llegar al final de la lista, terminando la simulación. Sino, al llegar al final de la lista se vuelve al inicio.

```
int KDS_sample(KDSource* kds, mcpl_particle_t* part);
```

Función de muestreo simple de partículas con una fuente `KDSource`. Redirige a `KDS_sample2`, con los argumentos `perturb=1`, `w_crit=1`, `bias=NULL` y `loop=1`.

```
double KDS_w_mean(KDSource* kds, int N, WeightFun bias);
```

Computar peso medio de las partículas en la lista utilizada por la fuente `kds`. Se utilizan `N` partículas para el cómputo. Si se fija `bias` distinto de `NULL`, se incluye la función de *weighting* `bias`.

```
void KDS_destroy(KDSource* kds);
```

Destruir fuente `KDSource`, liberando toda la memoria asociada.

```
typedef struct MultiSource{
    int len;     // Number of sources
    KDSource** s; // Array of sources
    double J;    // Total current [1/s]
    double* ws;  // Frequency weights of sources
```

```
    double* cdf; // cdf of sources weights
} MultiSource;
```

Estructura `MultiSource`. Modela un conjunto de fuentes KDE superpuestas. Los valores en el *array* `ws` definen las frecuencias de muestreo de cada fuente, mientras que sus intensidades se obtienen del parámetro `J` de cada fuente `KDSource`.

```
MultiSource* MS_create(int len, KDSource** s, const double* ws);
```

Crear estructura `MultiSource` en base a la cantidad de fuentes, el *array* de estructuras `KDSource`, y las frecuencias de muestreo deseadas. Durante la creación se computa la corriente total `J` y la función acumulativa de densidad `cdf`.

```
MultiSource* MS_open(int len, const char** xmlfilenames, const double* ws);
```

Cargar un conjunto de fuentes `KDSource` de los archivos de parámetros XML `xmlfilenames`, y construir estructura `MultiSource`.

```
int MS_sample2(MultiSource* ms, mcpl_particle_t* part, int perturb, double
    w_crit, WeightFun bias, int loop);
```

Función principal para muestreo de partículas con una fuente `MultiSource`. Se elige aleatoriamente una fuente usando las frecuencias definidas en `ms->ws`, y se le redirige el muestreo pasándole los mismos parámetros. Luego del muestreo se multiplica el peso de la partícula por el factor:

$$f_{SB} = \frac{J_i/J_{tot}}{w_i/w_{tot}} \tag{A.1}$$

Donde el subíndice i representa la fuente elegida para el muestreo, y *tot* la suma sobre todas las fuentes. De este modo se corrige la eventual discrepancia entre la intensidad relativa y la frecuencia relativa de muestreo, de acuerdo con la técnica de *source biasing*.

```
int MS_sample(MultiSource* ms, mcpl_particle_t* part);
```

Función de muestreo simple de partículas con una fuente `MultiSource`. Redirige a `MS_sample2`, con los argumentos `perturb=1`, `w_crit=1`, `bias=NULL` y `loop=1`.

```
double MS_w_mean(MultiSource* ms, int N, WeightFun bias);
```

Computar peso medio de las partículas en las listas de todas las fuentes. Computa los pesos medios de cada fuente mediante la función `KDS_w_mean` con los mismos parámetros `N` y `bias`, y computa el peso medio global como el promedio pesado mediante los valores en `ms->ws`.

```
void MS_destroy(MultiSource* ms);
```

Destruir fuente `MultiSource`, liberando toda la memoria asociada.

A.4.2. Estructura `PList`

La estructura `PList` modela una lista de partículas, actuando como *wrapper* de un archivo MCPL. Permite acceder a las partículas, e incluye la posibilidad de aplicarles una traslación y una rotación luego de la lectura.

A continuación se presentan las estructuras y funciones declaradas en el archivo `plist.h`.

```
typedef struct PList{
    char pt;                        // Particle type ("n", "p", "e", ...)
    int pdgcode;                    // PDG code for particle type

    char* filename;                 // Name of MCPL file
    mcpl_file_t file;               // MCPL file

    double* trasl;                  // PList translation
    double* rot;                    // PList rotation
    int x2z;                        // If true, apply permutation x,y,z -> y,z,x

    const mcpl_particle_t* part; // Pointer to selected particle
} PList;
```

Definición de la estructura `PList`. La misma tiene fijado el tipo de partícula según `pt`. Incluye la estructura correspondiente para la lectura de un archivo MCPL, además de (opcionalmente), parámetros que definen una transformación espacial a aplicar luego de la lectura de partículas. El parámetro `part` apunta a la ultima partícula leída, en todo momento.

```
PList* PList_create(char pt, const char* filename, const double* trasl, const
    double* rot, int switch_x2z);
```

Crear estructura `PList`. Se debe definir el tipo de partícula (`"n"` para neutrón, `"p"` para fotón, `"e"` para electrón) y el nombre del archivo MCPL. Opcionalmente se puede definir una traslación (*array* 3D) y una rotación (*array* 3D, formato eje-ángulo), o fijar en `NULL` dichos argumentos en caso contrario. La rotación se aplica luego de la traslación. Si `switch_x2z` es distinto de 0, luego de aplicar la rotación y traslación (de haberlas), se aplica la transformación $(x, y, z) \to (y, z, x)$.

```
int PList_get(const PList* plist, mcpl_particle_t* part);
```

Obtener partícula, aplicar transformaciones (de haberlas), y guardarla en `part`. La partícula se obtiene de `plist->part`, y no se modifica dicha variable luego de la lectura.

```
int PList_next(PList* plist, int loop);
```

Avanzar en la lista, actualizando la variable `plist->part`, hasta la siguiente partícula válida. Se considera una partícula como válida si tiene peso estadístico mayor a cero y su código PDG (tipo de partícula) coincide con el de la `PList`.

```
void PList_destroy(PList* plist);
```

Destruir estructura `PList`, liberando toda la memoria asociada.

A.4.3. Estructuras `Geometry` y `Metric`

La función principal de la estructura `Geometry` es perturbar partículas siguiendo la distribución del *kernel*. Lo logra redirigiendo dicha tarea a las métricas correspondientes a cada conjunto de variables, las cuales utilizan una función de perturbación específica

para cada tipo de métrica. `Geometry` también se encarga de administrar los anchos de banda y normalización de variables.

A continuación se presentan las definiciones, estructuras y funciones declaradas en `geom.h`.

```
typedef struct Metric Metric;

typedef int (*PerturbFun)(const Metric* metric, mcpl_particle_t* part,
    double bw);

struct Metric{
    int dim;            // Dimension
    float* scaling;     // Variables scaling
    PerturbFun perturb; // Perturbation function
    int nps;            // Number of metric parameters
    double* params;     // Metric parameters
};
```

Definición de la estructura `Metric`, en conjunto con la definición de función de perturbación. La función principal de dicha estructura es perturbar un conjunto de variables, lo cual realiza llamando a la función disponible en `perturb`, la cual a su vez utilizará, además del ancho de banda provisto como argumento, los escaleos de variables (`scaling`) y los parámetros de la métrica (`params`). El significado de dichos parámetros depende según el tipo de métrica, pudiendo representar tamaños de la fuente, valores mínimos, máximos o de referencia de variables, etc. Algunas métricas no poseen parámetros.

```
Metric* Metric_create(int dim, const double* scaling, PerturbFun perturb, int
    nps, const double* params);
```

Crear estructura `Metric`. Se debe proveer la dimensionalidad de la métrica (`dim`), los factores de escaleo (`scaling`), y la cantidad y valores de parámetros de la métrica (`nps` y `params`).

```
void Metric_destroy(Metric* metric);
```

Destruir estructura `Metric`, liberando toda la memoria asociada.

```
typedef struct Geometry{
    int ord;          // Number of submetrics
    Metric** ms;      // Submetrics
    char* bwfilename; // Bandwidth file name
    FILE* bwfile;     // Bandwidth file
    double bw;        // Normalized bandwidth

    double* trasl;    // Geometry translation
    double* rot;      // Geometry rotation
} Geometry;
```

Definición de la estructura `Geometry`. La misma contiene una cantidad `ord` de métricas almacenadas en `ms`. En el caso de que el ancho de banda sea adaptativo, `Geometry` administra la lectura del archivo donde se almacenan sus valores. En cualquier caso el parámetro `bw` contiene el ancho de banda actual. `KDSource` debe encargarse de que el ancho de banda presente en `bw` siempre se corresponda con la partícula presente en el parámetro `part` de la `PList`. Los parámetros `trasl` y `rot` representan la ubicación y orientación espacial de la fuente.

```
Geometry* Geom_create(int ord, Metric** metrics, double bw, const char*
    bwfilename, const double* trasl, const double* rot);
```

Crear estructura `Geometry`. Se debe indicar el orden `ord` de la misma (cantidad de métricas), proveer la ubicación de las métricas previamente creadas (`metrics`), y la posición y rotación de la fuente (`trasl` y `rot`). Para modelos con ancho de banda constante éste se debe proveer en el argumento `bw` y se debe fijar `bwfilename=NULL`, mientras que para ancho de banda adaptativo se debe indicar el archivo que contiene los anchos de banda (en formato binario de secuencia de puntos flotantes de simple precisión), y `bw` es ignorado.

```
int Geom_perturb(const Geometry* geom, mcpl_particle_t* part);
```

Perturbar partícula, siguiendo la distribución del *kernel* (*gaussiano*), y el ancho de banda presente en `geom->bw`.

```
int Geom_next(Geometry* geom, int loop);
```

Avanzar una posición en la lista de anchos de banda, en el caso de ancho de banda adaptativo. Para ancho de banda constante esta función no realiza ninguna acción.

```
void Geom_destroy(Geometry* geom);
```

Destruir estructura `Geometry`, liberando toda la memoria asociada.

```
#define E_MIN 1e-11
#define E_MAX 20
```

Valores mínimo y máximo de energía. Si luego de una perturbación se obtiene un valor de energía por fuera de este rango, se repite la perturbación.

```
int E_perturb(const Metric* metric, mcpl_particle_t* part, double bw);
```

Perturbar energía, con métrica simple de energía. En este caso `bw` multiplicado por el elemento de `scaling` correspondiente tiene unidades de energía (MeV).

```
int Let_perturb(const Metric* metric, mcpl_particle_t* part, double bw);
```

Perturbar energía, con métrica de letargía. En este caso `bw` multiplicado por el elemento de `scaling` correspondiente tiene unidades de letargía (adimensional).

```
int Vol_perturb(const Metric* metric, mcpl_particle_t* part, double bw);
```

Perturbar posición en sus 3 dimensiones, con métrica simple de posición. En este caso `bw` multiplicado por cada elemento de `scaling` correspondiente tiene unidades de posición (cm).

```
int SurfXY_perturb(const Metric* metric, mcpl_particle_t* part, double bw);
```

Perturbar posición en sus dimensiones x e y, con métrica simple de posición. En este caso `bw` multiplicado por cada elemento de `scaling` correspondiente tiene unidades de posición (cm).

```
int Guide_perturb(const Metric* metric, mcpl_particle_t* part, double bw);
```

Perturbar posición y dirección, con métrica de guía neutrónica. En este caso `bw` multiplicado por los dos primeros elementos de `scaling` tiene unidades de posición (cm), mientras que para los últimos dos tiene unidades de ángulo (grados). La métrica en dirección es polar, relativo a la normal de cada espejo. Internamente se transforma a las variables de guía z, t, θ, ϕ, se perturba dichas variables, y se antitransforma.

```
int Isotrop_perturb(const Metric* metric, mcpl_particle_t* part, double bw);
```

Perturbar dirección, con métrica isotrópica. En este caso `bw` multiplicado por el elemento de `scaling` correspondiente tiene unidades de ángulo (grados). La perturbación sigue la denominada distribución de Von Mises-Fischer.

```
int Polar_perturb(const Metric* metric, mcpl_particle_t* part, double bw);
```

Perturbar dirección, con métrica polar relativa a z. En este caso `bw` multiplicado por cada elemento de `scaling` correspondiente tiene unidades de ángulo (grados). Internamente se transforma a θ, ϕ, se perturba dichas variables, y se antitransforma.

```
int PolarMu_perturb(const Metric* metric, mcpl_particle_t* part, double bw);
```

Perturbar dirección, con métrica polar relativa a z, utilizando $\mu = cos(\theta)$. En este caso `bw` multiplicado por el primer elemento de `scaling` tiene unidades de μ (adimensional), mientras que por el segundo elemento tiene unidades de ángulo (grados). Internamente se transforma a μ, ϕ, se perturba dichas variables, y se antitransforma.

```
static const int _n_metrics = 8;
static const char* _metric_names[] = {"Energy", "Lethargy", "Vol", "SurfXY", "
    Guide", "Isotrop", "Polar", "PolarMu"};
static const PerturbFun _metric_perturbs[] = {E_perturb, Let_perturb,
    Vol_perturb, SurfXY_perturb, Guide_perturb, Isotrop_perturb, Polar_perturb,
     PolarMu_perturb};
```

Variables estáticas conteniendo la cantidad, nombres y funciones de perturbación de las métricas implementadas.

A.4.4. Utilidades generales

Además de las estructuras descritas previamente, se incluye un conjunto de utilidades generales para resolver problemas específicos. Esto incluye funciones matemáticas, conversión entre formatos de partícula y factores dosimétricos.

A continuación se describen las funciones declaradas en `utils.h`.

```
double rand_norm();
```

Obtener valor aleatorio con distribución normal, centrada en cero y con dispersión unitaria. Internamente utiliza el método de Box-Muller.

```
double *traslv(double *vect, const double *trasl, int inverse);
double *rotv(double *vect, const double *rotvec, int inverse);
```

Trasladar y rotar vector tridimensional. Realiza la transformación *in-place* sobre `vect` y lo retorna. Si `inverse` es distinto de 0 se aplica la transformación inversa. `trasl` es el vector de traslación, mientras que `rot` es el vector de rotación en formato ángulo-eje.

```
int pt2pdg(char pt);
char pdg2pt(int pdgcode);
```

Convertir de partícula en formato *char* (`"n"`: neutrón, `"p"`: fotón, `"e"`: electrón) a código PDG [21], y viceversa.

```
double interp(double x, const double *xs, const double *ys, int N);
```

Función de interpolación. Los *arrays* `xs` e `ys`, de longitud `N`, poseen los valores a interpolar. Se devuelve el valor interpolado en la posición `x`.

```
double H10_n_ARN(double E);
double H10_p_ARN(double E);
double H10_n_ICRP(double E);
double H10_p_ICRP(double E);
```

Factores dosimétricos en función de la energía, en unidades $[pSv\ cm^2]$. `n` indica neutrón y `p` indica fotón. `ARN` y `ICRP` indican la referencia para la tabla de interpolación que se utilizará. La interpolación se realiza en escala logarítmica.

Apéndice B

Estudio del error asociado a la fuente en simulaciones Monte Carlo

¿Cómo, cuándo y por qué es útil el método Kernel Density Estimation?

El objetivo de esta sección es responder esta pregunta, a través de un modelado matemático-estadístico de una simulación Monte Carlo.

B.1. Descripción del modelo de estudio

Se considerará una simulación Monte Carlo general con un único *tally*, cuyo resultado será denotado con T. Dos posibles fuentes serán consideradas, ambas basadas en una lista de partículas $X = \{x^{(i)}\}_{1<=i<=N}$ muestreadas de la distribución $S(x)$, siendo x el vector de fase de las partículas:

- **Fuente de *tracks***: Partículas muestreadas de la lista X, repitiéndose si es necesario. La distribución de fuente puede ser escrita en términos de deltas de Dirac:

$$S_t(x) = \frac{1}{N} \sum_i \delta\left(x - x^{(i)}\right) \tag{B.1}$$

- **Fuente KDE**: Partículas muestreadas de un modelo KDE construido con la lista X. Su ancho de banda se supondrá optimizado, y por simplicidad se lo considerará un único valor unidimensional h. La distribución de fuente tiene la expresión que define el método KDE:

$$S_{KDE}(x) = \frac{1}{Nh} \sum_i K\left(\frac{x - x^{(i)}}{h}\right) \tag{B.2}$$

En este apéndice se estudiarán los errores del resultado del *tally* T con cada fuente. Una de las herramientas estadísticas que se utilizará para cuantificar errores es el

carácter *poissoniano* del *tally*. En particular, se utilizará la regla según la cual si en un *tally* se cuentan n partículas, hay un error estadístico asociado de $\sqrt{n}$.

En cualquier simulación Monte Carlo, los *tallies* tienen dos tipos de errores:

- Error estadístico, el cual depende del número de partículas que alcanzaron la región del *tally* durante la simulación, el cual a su vez depende del número total de partículas simuladas.
- Error sistemático, el cual es constante durante la simulación. Incluye errores debidos a incertezas en la geometría, secciones eficaces, composiciones de materiales, distribución de fuente, etc.

Por el carácter *poissoniano* del proceso de *tallying*, su error estadístico puede ser estimado como parte del resultado, y también puede reducirse simplemente aumentando el número de partículas muestreadas en la simulación. Por el otro lado, los errores sistemáticos son más difíciles de cuantificar y controlar. En este estudio solo se considerará el error sistemático causado por errores en la distribución de la fuente de partículas.

Este apéndice tiene dos objetivos principales:

- Cuantificar el error sistemático en una simulación Monte Carlo debido a la fuente, para los dos tipos de fuente considerados. Para ello se proponen dos métodos:
  - Mediante la función importancia.
  - Mediante el método Total Monte Carlo.
- Utilizar las estimaciones del punto anterior para determinar la cantidad óptima de partículas de fuente, teniendo en cuenta los errores estadístico y sistemático, y la necesidad de minimizar los recursos computacionales.

B.2. Estimación del error sistemático debido a la fuente

B.2.1. Función importancia: *Tallies* como integrales pesadas de la distribución de fuente

Desde el punto de vista de la función importancia [40], también llamada flujo adjunto o función de Green, un resultado T de un *tally* puede ser expresado como:

$$T = \int I(x)S(x)dx \tag{B.3}$$

Donde I es la función importancia, y la integración es sobre todo el espacio de fases de la fuente. Como su nombre lo indica, I da la importancia relativa que cada vector

de fase tiene sobre el *tally*. Nótese que esta expresión analítica se corresponde con el valor obtenido para T en una simulación Monte Carlo en el límite de alto número de partículas simuladas, es decir error estadístico despreciable.

Fuente de *tracks*

Para la fuente de *tracks*, la expresión B.3 resulta:

$$T_t = \frac{1}{N}\sum_i I(x^{(i)}) \tag{B.4}$$

La varianza de T_t resulta entonces:

$$\begin{aligned} Var(T_t) &= \frac{1}{N^2} Var(N I_{mean}) \\ &= \frac{1}{N^2}(Var(N) I_{mean}^2 + N^2 Var(I_{mean})) \\ &= \frac{I_{mean}^2}{N} + \frac{1}{N} Var(I(x^{(i)}) \\ &= \frac{I_{mean}^2}{N} + \frac{1}{N}(\mathbb{E}I(x^{(i)})^2 - (\mathbb{E}I(x^{(i)}))^2) \\ &= \frac{I_{mean}^2}{N} + \frac{\sum_i I(x^{(i)})^2}{N^2} - \frac{I_{mean}^2}{N} \\ &= \frac{\sum_i I(x^{(i)})^2}{N^2} \end{aligned}$$

Donde se definió I_{mean} como la importancia media pesada por la distribución de fuente.

Finalmente, se obtiene el error sistemático relativo de T_t como la raíz cuadrada de la varianza:

$$\frac{\Delta T_t^{sist}}{T_t} = \frac{\sqrt{\sum_i I(x^{(i)})^2}}{\sum_i I(x^{(i)})} = \frac{1}{\sqrt{n_I^{eff}}} \tag{B.5}$$

Donde se definió el número efectivo de partículas importantes n_I^{eff} como:

$$n_I^{eff} = \frac{\left(\sum_i I(x^{(i)})\right)^2}{\sum_i I(x^{(i)})^2} \tag{B.6}$$

Nótese que, para funciones importancia con un pico bien definido, n_I^{eff} es aproximadamente la cantidad de partículas de fuente dentro de la región del pico de importancia. De ahí que, para la fuente de *tracks*, el error sistemático del *tally* asociado a la fuente equivale al error estadístico de la lista de *tracks* pesada por la función importancia.

Fuente KDE

Para la fuente KDE la expresión para el *tally* resulta:

$$\begin{aligned} T_{KDE} &= \int I(x) S_{KDE}(x) dx \\ &= \frac{1}{Nh} \int I(x) \sum_i K\left(\frac{x - x^{(i)}}{h}\right) dx \\ &= \frac{1}{Nh} \sum_i \int I(x) K\left(\frac{x - x^{(i)}}{h}\right) dx \end{aligned} \tag{B.7}$$

$$\tag{B.8}$$

Ante esta situación, se analizarán dos casos límite:

- $I(x)$ aproximadamente constante dentro del dominio[1] de cada $K\left(\frac{x-x^{(i)}}{h}\right)$. Esta situación se corresponde con funciones importancia suaves, usuales en problemas de moderación.
- Cada $K\left(\frac{x-x^{(i)}}{h}\right)$ aproximadamente constante dentro del dominio[1] de $I(x)$. Está situación se correspondo con funciones de importancia con un pico muy localizado y pronunciado, usuales en problemas de propagación en vacío (especialmente en la dependencia angular).

En el primer caso, la expresión para T_{KDE} resulta:

$$T_{KDE} = \frac{1}{N} \sum_i I(x^{(i)}) \int \frac{1}{h} K\left(\frac{x - x^{(i)}}{h}\right) dx = \frac{1}{N} \sum_i I(x^{(i)}) = T_t \tag{B.9}$$

Es decir que en este caso el método KDE resulta en el mismo resultado que la fuente de *tracks*.

Este resultado es esperable dado que, por la gran suavidad de I, el problema es poco sensible a perturbaciones de la fuente, como las que aplica la técnica KDE. Nótese que, dado que se espera que los *kernels* alrededor de cada muestra se superpongan, la condición que define este caso usualmente implica que el dominio de I cubrirá un número considerable de las partículas en la lista de *tracks*, es decir que el error de la fuente de *tracks* será relativamente bajo, reafirmando la conclusión de que, en esta situación, el método KDE no es necesario.

En la otra situación el *tally* T_{KDE} resulta:

$$T_{KDE} = \int I(x) dx S_{KDE}(x_0) = I_{tot} S_{KDE}(x_0) \tag{B.10}$$

[1] En este caso "dominio" se refiere a la región no nula o no despreciable de la función.

Donde x_0 es el punto central del pico de la función importancia.

Dado que en este caso T_{KDE} es proporcional a $S_{KDE}(x_0)$, sus errores relativos son idénticos. Utilizando una deducción equivalente a la realizada para la varianza de T_t se obtiene la varianza de $S_{KDE}(x_0)$. Considerando además el *bias* se tiene:

$$\frac{\Delta T^{sist}_{KDE}}{T_{KDE}} = \frac{\Delta S^{sist}_{KDE}}{S_{KDE}}(x_0) = \frac{\sqrt{\sum_i K\left(\frac{x_0 - x^{(i)}}{h}\right)^2}}{\sum_i K\left(\frac{x_0 - x^{(i)}}{h}\right)} + bias(x_0) = \frac{1}{\sqrt{n^{eff}_h(x_0)}} + bias(x_0) \tag{B.11}$$

Donde se definió $n^{eff}_h(x_0)$ de forma equivalente a n^{eff}_I, y se corresponde con el número efectivo de puntos utilizados por el método KDE en x_0, el cual equivale aproximadamente a la cantidad de puntos $x^{(i)}$ que cumplen $|x_0 - x^{(i)}| < h$. Cabe mencionar que el resultado obtenido para el error estadístico del método KDE coincide con lo presentado en [9], si se utiliza la siguiente aproximación:

$$n^{eff}_h(x_0) \approx \frac{\sum_i K\left(\frac{x_0 - x^{(i)}}{h}\right)}{R(K)} \tag{B.12}$$

Donde $R(K)$ es la rugosidad del *kernel*.

En situaciones usuales, la condición que define este caso implica que la cantidad de partículas en la región no nula (pico) de la función importancia será mucho menor que la cantidad de muestras dentro del dominio de un *kernel*. Es decir que $n^{eff}_h(x_0) >> n^{eff}_I$, y por lo tanto $1/\sqrt{n^{eff}_h(x_0)} << 1/\sqrt{n^{eff}_I}$. Por lo tanto, la utilización del método KDE reduce notablemente el error del *tally* asociado al error estadístico de la fuente, a expensas de un error adicional asociado al *bias* de suavizado.

Otra forma de verlo es que, debido al comportamiento de los métodos de optimización de ancho de banda, esta situación puede interpretada como una en la cual n^{eff}_I es bajo. Si el número de partículas que el *tally* "mira" es muy bajo, el error asociado será muy grande, y por lo tanto es razonable que sea preferible promediar sobre una región un poco mayor, aún cuando esto traiga un sesgo adicional.

Si bien no es posible, de forma general, determinar si la reducción en el error estadístico supera el error añadido por el *bias*, esta es sin dudas la situación en la cual KDE puede ser útil. Además, los métodos de optimización de ancho de banda deberían optimizar el balance entre varianza y *bias*, apoyando la hipótesis de que el método KDE puede dar mejores resultados que la fuente de *tracks*.

B.2.2. Estimación de errores con Total Monte Carlo

En una simulación real, la función de importancia es desconocida, por lo cual las expresiones previamente obtenidas no pueden ser evaluadas. Puede ser útil entonces

aplicar el método Total Monte Carlo [41] relativo a la fuente. Esto significa dividir la lista de partículas X en un número de subconjuntos s (100, por ejemplo), y usar cada uno para construir ambos tipos de fuentes en una simulación. Para cada fuente, se obtendrá entonces un conjunto de $\{T_1, ..., T_s\}$ de resultados para el *tally*, con dispersión σ_T. El resultado final es el promedio de dichos valores, y por lo tanto el error asociado a la dispersión estadística de cada fuente resulta:

$$\frac{\Delta T^{TMC}}{T} = \frac{\sigma_T}{\sqrt{s}} \tag{B.13}$$

Nótese que esta expresión asume un error estadístico del *tally* despreciable.

Este método no provee una estimación del error asociado al *bias* de la fuente KDE, pero este podría asumirse como del mismo orden de magnitud que el causado por el error estadístico. También podría cuantificarse comparando los resultados con ambas fuentes, en el caso de que el error estadístico con la fuente de *tracks* sea razonablemente bajo.

B.3. Multiplicación del número de partículas de fuente

En una simulación real, el valor obtenido del *tally* tiene tanto error estadístico como sistemático. El error total usualmente se toma como:

$$\frac{\Delta T}{T} = \sqrt{\left(\frac{\Delta T^{sist}}{T}\right)^2 + \left(\frac{\Delta T^{stat}}{T}(M)\right)^2} \tag{B.14}$$

Donde M es el número total de partículas simuladas.

A medida que M aumenta, el error estadístico decrece. El sistemático, por el contrario, permanece constante. Mientras el error estadístico sea predominante, un número mayor de partículas simuladas se traduce en una reducción significativa del error total. En la situación inversa, producir nuevas partículas, con el costo computacional que implica, no resulta de gran utilidad pues el error total se mantendrá aproximadamente constante e igual a la componente sistemática. Un punto crítico razonable para distinguir estas situaciones es el punto en el que ambos errores se igualan. El objetivo de esta sección será obtener expresiones para el M óptimo tal que:

$$\frac{\Delta T^{stat}}{T}(M_{opt}) = \frac{\Delta T^{sist}}{T} \tag{B.15}$$

Se definirá además el factor de multiplicación del número de partículas de fuente $k = M/N$.

El error estadístico de un *tally* puede expresarse como:

$$\frac{\Delta T^{stat}}{T} = \frac{1}{\sqrt{n_R}} \tag{B.16}$$

Donde n_R es el número de partículas que alcanzaron el *tally*. En una simulación con M partículas de fuente n_R puede expresarse como:

$$n_R = p_T M = k p_I n_I^{eff} \tag{B.17}$$

Donde p_T es la probabilidad de que una partícula de fuente alcance el *tally*. Se expresó dicha probabilidad como $p_T = (n_I^{eff}/N)p_I$, donde p_I se interpreta como la probabilidad de que una partícula de fuente, cuyo vector de fase se sitúa en la región del pico de la función importancia, alcance el *tally*. En una simulación en vacío usualmente se tiene $p_I \approx 1$.

La expresión B.16 puede igualarse a cualquiera de las expresiones para el error sistemático obtenidas anteriormente, tanto a través de la función importancia como con Total Monte Carlo, y utilizarse para despejar el M o k óptimo. Nótese que el valor de p_T puede estimarse fácilmente con una simulación de prueba. De todos modos, a continuación se despejan algunas expresiones en términos de la función importancia.

Para la fuente de *tracks*, combinando B.5, B.16 y B.17 se obtiene un factor de multiplicación óptimo:

$$k_{opt} = \frac{1}{p_I} \tag{B.18}$$

Lo cual significa que en simulaciones con blindajes o moderadores, entre la fuente y el detector, puede ser útil repetir las partículas en la lista de *tracks*, pero en un problema de propagación en vacío esto no resulta beneficioso.

Para la fuente KDE, en el caso de función de importancia muy localizada, combinando B.11, B.16 y B.17 se obtiene:

$$k_{opt} = \frac{1}{n_I^{eff} p_I \left(\frac{1}{\sqrt{n_h^{eff}}} + bias(x_0) \right)^2} \tag{B.19}$$

Despreciando el *bias* se obtiene:

$$k_{opt} = \frac{n_h^{eff}}{n_I^{eff}} \frac{1}{p_I} \tag{B.20}$$

Esto quiere decir que, gracias al método KDE, el número óptimo de partículas simuladas se incrementa en un factor n_h^{eff}/n_I^{eff}, lo cual significa una reducción en el error total del *tally* de un factor $\sqrt{n_I^{eff}/n_h^{eff}}$.

Bibliografía

[1] Abbate, O. I. Cálculo de blindajes asociado a guías de neutrones. Proyecto Fin de Carrera, Instituto Balseiro, 6 2017. 2, 4, 45, 65, 66

[2] Fairhurst, R. E. Cálculo neutrónico detallado de haces y guías de neutrones del reactor RA-10. Proyecto Fin de Carrera, Instituto Balseiro, 6 2017. 2, 4, 45, 48, 65, 69

[3] Ayala, J. E. Implementación de una línea de cálculo basada en el código Tripoli a problemas de blindaje del reactor RA-10. Proyecto Fin de Carrera, Instituto Balseiro, 6 2019. 2, 4

[4] Lefmann, K., Nielsen, K. Mcstas, a general software package for neutronray-tracing simulations. *Neutron News*, **10** (3), 20–23, 1999. URL https://doi.org/10.1080/10448639908233684. 5, 27

[5] Willendrup, P., Farhi, E., Lefmann, K. Mcstas 1.7 - a new version of the flexible monte carlo neutron scattering package. *Physica B: Condensed Matter*, **350** (1, Supplement), E735–E737, 2004. URL https://doi.org/10.1016/j.physb.2004.03.193, proceedings of the Third European Conference on Neutron Scattering.

[6] Willendrup, P., Farhi, E., Knudsen, E., Filges, U., Lefmann, K. Mcstas: Past, present and future. *Journal of Neutron Research*, **17** (1), 35 – 43, 2014. URL https://doi.org/10.3233/JNR-130004. 5, 27

[7] Brun, E., Damian, F., Diop, C., Dumonteil, E., Hugot, F., Jouanne, C., *et al.* Tripoli-4®, cea, edf and areva reference monte carlo code. *Annals of Nuclear Energy*, **82**, 151–160, 2015. URL https://doi.org/10.1016/j.anucene.2014.07.053, Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms. 6, 27

[8] Scott, D. W. Multivariate density estimation: theory, practice, and visualization. John Wiley & Sons, 2015. 9, 28

[9] Hansen, B. E. Lecture notes on nonparametrics. University of Wisconsin, 2009. [Online]. Available at: https://www.ssc.wisc.edu/~bhansen/718/NonParametrics1.pdf. [Accesed: 5-nov-2021]. 12, 115

[10] Weinberger, K. Machine learning for intelligent systems, cap. 16: KD Trees. Cornell University, 2018. [Online]. Available at: http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote16.html. [Accesed: 5-nov-2021]. 13, 24, 29

[11] Turlach, B. Bandwidth selection in kernel density estimation: A review. *Handbook of Systemic Autoimmune Diseases*, 02 1999. [Online]. Available at: https://www.researchgate.net/publication/2316108_Bandwidth_Selection_in_Kernel_Density_Estimation_A_Review. [Accesed: 5-nov-2021]. 14

[12] Silverman, B. Density Estimation for Statistics and Data Analysis (1st ed.). Routledge, 1998. URL https://doi.org/10.1201/9781315140919. 15, 28, 34

[13] Arlot, S., Celisse, A. A survey of cross-validation procedures for model selection. *Statistics Surveys*, **4**, Jan 2010. URL http://doi.org/10.1214/09-SS054. 15, 28

[14] Duin, R. On the choice of smoothing parameters for parzen estimators of probability density functions. *IEEE Transactions on Computers*, **C-25** (11), 1175–1179, 1976. URL http://doi.org/10.1109/TC.1976.1674577. 16, 28, 34

[15] Terrell, G. R., Scott, D. W. Variable kernel density estimation. *The Annals of Statistics*, **20** (3), 1236 – 1265, 1992. URL https://doi.org/10.1214/aos/1176348768. 17, 34

[16] Papamakarios, G. Neural density estimation and likelihood-free inference. arXiv:1910.13233, 2019. URL https://arxiv.org/abs/1910.13233. 18, 20

[17] Germain, M., Gregor, K., Murray, I., Larochelle, H. Made: Masked autoencoder for distribution estimation. arXiv:1502.03509, 2015. URL https://arxiv.org/abs/1502.03509. 19

[18] Papamakarios, G., Pavlakou, T., Murray, I. Masked autoregressive flow for density estimation. arXiv:1705.07057, 2018. URL https://arxiv.org/abs/1705.07057. 20

[19] Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., Welling, M. Improving variational inference with inverse autoregressive flow. arXiv:1606.04934, 2017. URL https://arxiv.org/abs/1606.04934. 20, 24

[20] T. Kittelmann, e. a. Monte Carlo Particle Lists: MCPL. *Computer Physics Communications*, **218**, 17–42, 2017. URL https://doi.org/10.1016/j.cpc.2017.04.012. 26, 31

[21] K. A. Olive, e. Review of particle physics. *Chinese Physics C*, **38** (9), 2014. URL https://doi.org/10.1088/1674-1137/38/9/090001. 26, 101, 109

[22] Pelowitz, D. MCNPX Users Manual Version 2.7.0, 2011. Report LA-CP-11-00438. 27

[23] Team, X.-. M. C. MCNP - Version 5, Vol. I: Overview and Theory, 2003. Report LA-UR-03-1987.

[24] Werner, C. MCNP Users Manual - Code Version 6.2, 2017. Report LA-UR-17-29981. 27

[25] Sato, T., Iwamoto, Y., Hashimoto, S., Ogawa, T., Furuta, T., ichiro Abe, S., *et al.* Features of Particle and Heavy Ion Transport code System (PHITS) version 3.02. *Journal of Nuclear Science and Technology*, **55** (6), 684–690, 2018. URL https://doi.org/10.1080/00223131.2017.1419890. 27

[26] Agostinelli, S., Allison, J., Amako, K., Apostolakis, J., Araujo, H., Arce, P., *et al.* Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **506** (3), 250–303, 2003. URL https://doi.org/10.1016/S0168-9002(03)01368-8. 27

[27] Allison, J., Amako, K., Apostolakis, J., Araujo, H., Arce Dubois, P., Asai, M., *et al.* Geant4 developments and applications. *IEEE Transactions on Nuclear Science*, **53** (1), 270–278, 2006. URL https://doi.org/10.1109/TNS.2006.869826.

[28] Allison, J., Amako, K., Apostolakis, J., Arce, P., Asai, M., Aso, T., *et al.* Recent developments in Geant4. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **835**, 186–225, 2016. URL https://doi.org/10.1016/j.nima.2016.06.125. 27

[29] Bergbäck Knudsen, E., Prodi, A., Baltser, J., Thomsen, M., Willendrup, P., Sanchez Del Rio, M., *et al.* McXtrace: A Monte Carlo software package for simulating X-ray optics, beamlines and experiments. *Journal of Applied Crystallography*, **46** (3), 679–696, 2013. URL https://doi.org/10.1107/S0021889813007991. 27

[30] Jones, E., Oliphant, T., Peterson, P., *et al.* SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/. 28

[31] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., *et al.* Scikit-learn: Machine learning in Python, 2007. URL https://scikit-learn.org. 28

[32] Seabold, S., Perktold, J. statsmodels: Econometric and statistical modeling with python, 2010. URL https://www.statsmodels.org/. 28

[33] tommyod. Kdepy, 2018. URL https://kdepy.readthedocs.io/en/latest/. 28, 30

[34] Gramacki, A., Gramacki, J. FFT-based fast computation of multivariate kernel density estimators with unconstrained bandwidth matrices. *Journal of Computational and Graphical Statistics*, **26** (2), 459–462, 2017. URL https://doi.org/10.1080/10618600.2016.1182918. 29

[35] Jakob, W. Numerically stable sampling of the von mises fisher distribution on S^2 (and other tricks). Realistic Graphics Lab, 2012. [Online]. Available at: https://www.mitsuba-renderer.org/~wenzel/files/vmf.pdf. [Accesed: 5-nov-2021]. 32

[36] Kolevatov, R., Schanzer, C., Boeni, P. Neutron absorption in supermirror coatings: Effects on shielding. arXiv:1708.04486, 2018. URL https://arxiv.org/abs/1708.04486. 37, 66

[37] IAEA. Live chart of nuclides — IAEA Nuclear Data Services. [Online]. Available at: https://www-nds.iaea.org/relnsd/vcharthtml/VChartHTML.html. [Accesed: 5-nov-2021]. 38, 100

[38] Aygün, B. Neutron and gamma radiation shielding properties of high-temperature-resistant heavy concretes including chromite and wolframite. *Journal of Radiation Research and Applied Sciences*, **12** (1), 352–359, 2019. URL https://doi.org/10.1080/16878507.2019.1672312. 49

[39] IAEA. Prompt gamma activation analysis database files. [Online]. Available at: https://www-nds.iaea.org/pgaa. [Accesed: 5-nov-2021]. 67

[40] G. I. Bell, S. G. Nuclear Reactor Theory. Van Nostrand Reinhold Co., New York, 1970. 112

[41] Shintaro Hashimoto, T. S. Estimation method of systematic uncertainties in monte carlo particle transport simulation based on analysis of variance. *Journal of Nuclear Science and Technology*, 2019. URL https://doi.org/10.1080/00223131.2019.1585989. 116

Agradecimientos

Nuevamente tengo que agradecer a mis padres de tesis Ariel y Nacho, quienes me siguen acompañando y enseñando. Agradezco también al recientemente formado grupo de fuentes de distribuciones, Norbert, Zoe, José y Javier, gracias al cual esta línea de investigación tomó mucha mejor forma y se volvió menos solitaria.

Agradezco a Anto por bancarme todo este tiempo en casa, que se volvió el lugar de trabajo, y seguir acompañándome en este aprendizaje de adultez que llevamos juntes (y con nuestros gatitos). Agradezco a mi familia, que me dio muchos lindas escapaditas a Bolsón durante esta pandemia.

Finalmente, reafirmo la importancia del estado y la educación pública, sin la cual todo esto no sería posible.