



ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

**MODELOS Y MÉTODOS DE RECONSTRUCCIÓN DE IMÁGENES
MÉDICAS**

**RECONSTRUCCIÓN DE IMÁGENES DE TOMOGRAFÍA POR EMISIÓN
DE POSITRONES**

**TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA
LA OBTENCIÓN DEL TÍTULO DE MATEMÁTICO**

DANIEL MATÍAS LARA NARANJO

daniel.lara01@epn.edu.ec

DIRECTORA: EVELYN GABRIELA CUEVA JARAMILLO, PH. D.

evelyn.cuevaj@epn.edu.ec

QUITO, 4 DE AGOSTO DE 2024

CERTIFICACIONES

Yo, DANIEL MATÍAS LARA NARANJO, declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Daniel Matías Lara Naranjo

Certifico que el presente trabajo de integración curricular fue desarrollado por Daniel Matías Lara Naranjo, bajo mi supervisión.

Evelyn Gabriela Cueva Jaramillo, Ph. D.

DIRECTORA

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el producto resultante del mismo, es público y estará a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Daniel Matías Lara Naranjo

Evelyn Gabriela Cueva Jaramillo, Ph. D.

RESUMEN

Las imágenes médicas se han convertido en una herramienta fundamental para el diagnóstico y tratamiento de enfermedades apoyadas en diversos fenómenos físicos. En este trabajo se presenta un estudio sobre las técnicas de reconstrucción de imágenes por emisión de positrones (PET).

Primero, se realizará una breve introducción a los problemas inversos, la metodología para el procesamiento de imágenes y los conceptos relacionados con las transformadas de Fourier y Radón. A continuación, se presentará la teoría física subyacente a las imágenes de tomografía computarizada (CT), la cual servirá como base para estudiar la tomografía por emisión de positrones (PET).

Una vez establecida la base del problema, nos enfocaremos en estudiar las metodologías numéricas que permiten la reconstrucción de imágenes a partir de los datos obtenidos durante el proceso de adquisición. Presentaremos el método analítico de retroproyección filtrada (FBP), así como los algoritmos ML-EM (Maximum Likelihood Expectation Maximization) y OSEM (Ordered Subsets Expectation Maximization) para PET.

Finalmente, se presentan los resultados de las reconstrucciones experimentales obtenidas a partir de datos reales recuperados de fuentes abiertas, utilizando las librerías CIL y NiftyPET para diversos casos de estudio.

Palabras clave: problemas inversos, reconstrucción de imágenes, tomografía computarizada, tomografía por emisión de positrones, optimización numérica, PDHG, OSEM, ML-EM, NiftyPET, CIL.

ABSTRACT

Medical images have become a fundamental tool for the diagnosis and treatment of diseases supported by various physical phenomena. In this work, we present a study on reconstruction techniques for positron emission tomography (PET) images.

First, we will present a brief review of inverse problems, the methodology for image processing, and concepts related to the Fourier and Radon transforms that we will use later for the subsequent mathematical modeling. Then, we will present the physical theory behind computed tomography (CT) images as a basis for studying PET.

Once the base of the problem is presented, we will focus on the study of numerical methodologies that allow us to reconstruct the images from the data obtained in the acquisition process. For this, we will present the analytical method of filtered back-projection (FBP) and the Maximum Likelihood Expectation Maximization (ML-EM) and Ordered Subsets Expectation Maximization (OSEM) algorithms for PET.

Finally, experimental reconstruction results obtained from real data recovered from open sources are presented using the CIL and NiftyPET libraries for different case studies.

Keywords: inverse problems, image reconstruction, computed tomography, positron emission tomography, numeric optimization, PDHG, OSEM, ML-EM, NiftyPET, CIL.

DEDICATORIA

*A mi madre,
por su apoyo incondicional
en cada locura que se me ha ocurrido.*

AGRADECIMIENTOS

A mi madre, por su todo su amor y apoyo desde mis primeros años en la escuela hasta el día en el que escribo estas líneas.

A mis abuelitos, pues sin su apoyo y complicidad no hubiera logrado todo lo que he alcanzado hasta ahora.

A Lucía, mi tía, pues sin su consejo nunca hubiera conocido la existencia de la carrera que hoy estoy por finalizar.

A mi familia que de una u otra manera siempre ha estado presente en mi vida y han hecho de esta una experiencia más amena y maravillosa.

A Namie, por ser mi compañera en estos últimos años y haberme apoyado en todo momento.

A Andrés, por pasar de ser un profesor a un amigo y guía que en más de una ocasión me ha sacado de apuros en temas que no lograba entender.

A Evelyn, por darme la apertura a un área maravillosa que desconocía de su existencia y haberme guiado en este último año de mi vida académica, haciendo de nuestras reuniones un espacio de aprendizaje y crecimiento.

A mis amigos; Gabriela y Luis, por tantos deberes, tareas, almuerzos, risas y tristezas compartidas en estos años. Karen y Mayerlin, amigas que conocí en el colegio y aún hoy compartimos amenas reuniones y conversaciones. Daniel, por hacer de las largas horas en la asociación algo más ameno y divertido.

A todos los que me adoptaron cuando fui a la asociación en primer semestre y supieron guiarme con su experiencia y brindarme su amistad hasta el día de hoy.

Finalmente, a todas aquellas personas que a lo largo de estos años han sido parte de mi vida y para las cuales el espacio en el margen de esta hoja no es suficiente para enumerarlas.

Índice general

| | |
|---|----------|
| 1. Descripción del componente desarrollado | 1 |
| 1.1. Objetivo general | 1 |
| 1.2. Objetivos específicos | 2 |
| 1.3. Alcance | 2 |
| 2. Marco teórico | 3 |
| 2.1. Problemas Inversos | 3 |
| 2.1.1. Problemas mal condicionados | 4 |
| 2.1.2. Crímenes inversos | 5 |
| 2.2. Procesamiento básico de imágenes | 6 |
| 2.2.1. Imágenes e histogramas | 8 |
| 2.2.2. Ecualización de histogramas | 8 |
| 2.2.3. Detección de bordes | 8 |
| 2.3. Transformada de Fourier | 10 |
| 2.3.1. Espacio de Schwartz | 11 |
| 2.3.2. Transformada de Fourier inversa | 12 |
| 2.4. Transformada de Radón | 13 |
| 2.4.1. Teorema del corte central | 17 |

| | | |
|-----------|---|-----------|
| 2.5. | Nociones básicas de convexidad | 20 |
| 2.5.1. | Métodos de punto de silla - Algoritmos tipo Primal-Dual | 22 |
| 2.6. | Introducción a la optimización continua para imágenes | 25 |
| 2.6.1. | Métricas | 25 |
| 2.6.2. | Restauración de imágenes | 27 |
| 2.7. | Tomografía por Emisión de Positrones | 32 |
| 2.7.1. | Breve historia de las imágenes médicas | 32 |
| 2.7.2. | Introducción a la tomografía por emisión de positrones | 33 |
| 2.7.3. | Tipos de eventos | 35 |
| 2.7.4. | Reconstrucción de imágenes PET | 37 |
| 2.7.5. | Ejemplos prácticos de uso médico | 50 |
| 3. | Experimentación | 53 |
| 3.1. | Hardware y software | 53 |
| 3.1.1. | Datos | 54 |
| 3.2. | Core Image Libray (CIL) | 55 |
| 3.2.1. | Definición de geometrías | 55 |
| 3.2.2. | Reconstrucción a partir de datos sin procesar | 57 |
| 3.2.3. | Preprocesamiento de datos | 60 |
| 3.2.4. | Reconstrucción de imágenes usando PDHG | 67 |
| 3.3. | NiftyPET | 74 |
| 3.3.1. | Carga de los datos | 74 |
| 3.3.2. | Mapas de atenuación | 76 |
| 3.3.3. | Sinogramas | 78 |
| 3.3.4. | Reconstrucción | 79 |
| 4. | Conclusiones y Recomendaciones | 89 |

| | |
|---|------------|
| 4.1. Conclusiones | 89 |
| 4.2. Recomendaciones | 90 |
| A. Anexos | 91 |
| A.1. Notación | 91 |
| A.2. Códigos | 92 |
| A.2.1. Experimentación con CIL | 103 |
| A.3. Experimentación con NiftyPET | 115 |
| A.4. Tablas | 117 |
| Bibliografía | 118 |

Índice de figuras

| | |
|---|----|
| 2.1. Ejemplo de un problema directo e inverso. | 4 |
| 2.2. Icono cámara en diferentes resoluciones. | 6 |
| 2.3. Imagen original. | 7 |
| 2.4. Imagen original junto con su histograma y función de distribución acumulada. | 8 |
| 2.5. Ecualización de histogramas. | 9 |
| 2.6. Detección de bordes. | 9 |
| 2.7. Representación de la transformada de Radón. Adaptado de [26]. | 14 |
| 2.8. Progreso del algoritmo FBP para la reconstrucción de la Figura 2.20. | 17 |
| 2.9. Representación de la aplicación del teorema del Teorema 2.3. Adaptado de [26]. | 19 |
| 2.10. Ejemplo de eliminación de ruido en una imagen digital usando el modelo ROF | 29 |
| 2.11. Ejemplo de eliminación de ruido en una imagen digital usando el modelo ROF | 30 |
| 2.12. Ejemplo de eliminación de ruido en una imagen digital con ruido de tipo sal y pimienta usando el modelo $TV - \ell_1$ | 31 |
| 2.13. Detalle de la imagen restaurada en la Figura 2.12c. | 32 |
| 2.14. Primera imagen de rayos X de la mano de la esposa de Röntgen. | 33 |

| | |
|--|----|
| 2.15. Imagen de tomografía computarizada de un cráneo. | 33 |
| 2.16. Ejemplo de emisión de fotones. Imagen tomada de [26] Pág. 209. | 35 |
| 2.17. Ventana de coincidencia para dos eventos. | 36 |
| 2.18. Clasificación de eventos en una imagen PET. | 37 |
| 2.19. Parametrización de un sinograma. | 39 |
| 2.20. Ejemplo de un sinograma para un círculo. | 40 |
| 2.21. Ejemplo de un sinograma para una figura mixta. | 41 |
| 2.22. Ejemplo de un sinograma para la forma de Shepp-Logan. | 41 |
| 2.23. Esquema de un escáner PET con múltiples anillos de detectores. | 42 |
| 2.24. Ejemplo datos 2D multi-corte | 43 |
| 2.25. Ejemplo visualización 3D de datos 2D multi-corte | 43 |
| 2.26. Ejemplo de una imagen discreta. | 46 |
| 2.27. Imágenes PET/CT mostrando hipometabolismo en el lóbulo temporal izquierdo. Imágenes obtenidas de [16]. | 51 |
| 2.28. Imágenes PET/MR de un paciente con glioma en el tálamo izquierdo. Imágenes obtenidas de [16]. | 52 |
| 2.28. Imágenes PET/MR de un paciente con glioma en el tálamo izquierdo. Imágenes obtenidas de [16]. | 52 |
| 3.1. Representación 3D de una geometría abanico 2D. | 56 |
| 3.2. Representación 3D de una geometría cono 3D. | 56 |
| 3.3. Geometría 2D. | 57 |
| 3.4. Geometría 3D. | 58 |
| 3.5. Geometría de abanico 2D. | 58 |
| 3.6. Geometría de cono 3D. | 59 |
| 3.7. Geometría de los datos sin procesar. | 59 |
| 3.8. Sinograma de los datos sin procesar. | 60 |
| 3.9. Proyecciones de los datos sin procesar. | 61 |

| | |
|--|----|
| 3.10. Reconstrucción FDK. | 61 |
| 3.11. Geometría de los datos para preprocesamiento. | 62 |
| 3.12. Proyección de los datos sin procesar. | 63 |
| 3.13. Proyección de los datos convertidos a absorción. | 63 |
| 3.14. Proyección de los datos corregidos. | 64 |
| 3.15. Proyección de los datos con artefactos. | 65 |
| 3.16. Comparativa de reconstrucción FBP con artefactos. | 65 |
| 3.17. Máscara de los datos con artefactos. | 66 |
| 3.18. Comparativa de sinogramas con y sin máscara. | 66 |
| 3.19. Comparativa de reconstrucción FBP con artefactos corregidos. | 67 |
| 3.20. Geometría de los datos para PDHG. | 69 |
| 3.21. Sinograma y proyección de los datos para PDHG. | 69 |
| 3.22. Reconstrucción FBP. | 70 |
| 3.23. Comparativa reconstrucción con regularización L_1 vs FBP. | 71 |
| 3.24. Reconstrucción con regularización de variación total. | 73 |
| 3.25. Reconstrucción con regularización de Tykhonov. | 74 |
| 3.26. Reconstrucciones con PDHG usando diferentes métodos de regularización. | 75 |
| 3.27. Estructura de directorios para los datos de PET. | 76 |
| 3.28. Mapas de atenuación con corte axial. | 77 |
| 3.29. Mapas de atenuación con diferentes cortes. | 78 |
| 3.30. Sinogramas PET. | 79 |
| 3.31. Conteo de eventos en los sinogramas. | 79 |
| 3.32. Evolución de la métrica SSIM en función del número de iteraciones. | 80 |
| 3.33. Reconstrucción ML-EM corte axial. | 81 |
| 3.34. Reconstrucción ML-EM corte sagital. | 82 |
| 3.35. Reconstrucción ML-EM con 50 iteraciones. | 82 |

| | |
|---|----|
| 3.36. Tiempos de ejecución acumulados por iteración ML-EM. | 83 |
| 3.37. SSIM por iteración ML-EM. | 83 |
| 3.38. Reconstrucción OSEM corte axial. | 84 |
| 3.39. Reconstrucción OSEM corte sagital. | 84 |
| 3.40. Reconstrucción OSEM corte coronal. | 85 |
| 3.41. Tiempos de ejecución acumulados por iteración OSEM. | 85 |
| 3.42. Métrica SSIM por iteración OSEM. | 86 |
| 3.43. Diferencia relativa respecto a la imagen de referencia por iteración OSEM con la métrica SSIM. | 86 |
| 3.44. Comparativa de resultados ML-EM y OSEM. | 87 |
| 3.44. Comparativa de resultados ML-EM y OSEM. | 88 |
| 3.44. Comparativa de resultados ML-EM y OSEM. | 88 |

Índice de tablas

| | |
|---|-----|
| 2.1. Valores de los píxeles de la imagen de la Figura 2.2a. | 7 |
| 2.2. Tiempo de ejecución promedio en la eliminación de ruido en una imagen digital. | 28 |
| 2.3. Tiempo de ejecución promedio en la eliminación de ruido en una imagen digital. | 30 |
| 2.4. Tiempo de ejecución promedio en la eliminación de ruido en una imagen digital. | 31 |
| 3.1. Características de las computadoras utilizadas en la experimentación. . . . | 53 |
| 3.2. Librerías utilizadas en la experimentación. | 54 |
| 3.3. Datos utilizados en la experimentación. | 54 |
| 3.4. Comparación de tiempos de ejecución de los métodos de regularización. . | 74 |
| 3.5. Tiempos de ejecución. | 82 |
| 3.6. Detalles de la reconstrucción OSEM. | 87 |
| A.1. Resultados de la experimentación para ML-EM | 117 |

Capítulo 1

Descripción del componente desarrollado

Este proyecto revisa los conceptos esenciales para abordar la reconstrucción de imágenes por emisión de positrones. Para ello, se llevará a cabo una revisión general de los problemas inversos, el procesamiento de imágenes, los fenómenos físicos y los algoritmos de optimización necesarios para la reconstrucción de imágenes de Tomografía por emisión de positrones (PET). Además, como paso intermedio, se examinarán brevemente casos de estudio relacionados con tomografía computarizada (CT), lo que proporcionará una mejor comprensión para la experimentación.

Una vez descrita la teoría necesaria, se llevarán a cabo experimentos numéricos utilizando los módulos CIL y NiftyPET para la reconstrucción de imágenes CT y PET, respectivamente. En esta sección, se analizará la calidad de las imágenes obtenidas a partir de diferentes conjuntos de datos y métodos de reconstrucción, evaluando su desempeño y comparando los resultados obtenidos.

1.1. Objetivo general

Estudiar la reconstrucción de imágenes generadas por un escáner de tomografía por emisión de positrones (PET), aplicando modelos físicos y matemáticos adecuados para lograr una reconstrucción significativa.

1.2. Objetivos específicos

1. Enunciar los métodos utilizados para el procesamiento de imágenes digitales, así como los específicos para la reconstrucción de imágenes médicas.
2. Explicar el modelo físico subyacente en el que se basa el escáner de tomografía por emisión de positrones (PET).
3. Estudiar los fundamentos matemáticos que respaldan la reconstrucción de imágenes por emisión de positrones (PET).
4. Comprender el funcionamiento y la reconstrucción de imágenes en tomografía computarizada (CT) como base para la reconstrucción de imágenes en tomografía por emisión de positrones (PET).
5. Implementar la reconstrucción de imágenes PET utilizando el módulo de Python NiftyPET para llevar a cabo diversos ensayos numéricos.

1.3. Alcance

El alcance de este trabajo es la revisión, comprensión y realización de experimentos sobre la reconstrucción de imágenes PET. Como una primera aproximación, se describirán los problemas inversos y *mal puestos*. Luego, se explicarán brevemente las metodologías básicas para el procesamiento de imágenes a partir de lo mencionado en [26, 6, 4]; así como herramientas útiles tales como la Transformada de Fourier [18] y la Transformada de Radón [33].

Una vez presentados los fundamentos, estudiaremos el funcionamiento de un escáner PET y los modelos matemáticos que lo sustentan, basados en [1]. Finalmente, realizaremos ensayos numéricos para la reconstrucción de imágenes CT utilizando el módulo *Core Imaging Library* (CIL) y para PET empleando el módulo NiftyPET, todo ello implementado en lenguaje Python.

Capítulo 2

Marco teórico

2.1. Problemas Inversos

Para poder definir un *problema inverso* consideremos la definición de un *problema directo*. De manera intuitiva un problema inverso es aquel en el que se busca encontrar el efecto a partir de la causa. Para una mejor comprensión, consideremos el siguiente ejemplo¹ que nos permite ilustrar un problema directo.

Ejemplo 2.1. *Supongamos que estamos interesados en determinar la temperatura del aire a nuestro alrededor. Para este fin podemos emplear un termómetro, este nos indica aprovechando el hecho de que el mercurio se expande con el aumento de la temperatura. Así, el modelo directo considerado es el que describe la relación entre la temperatura y la expansión del mercurio. Por lo tanto, estamos observando el efecto directo (la expansión del mercurio) de la causa (la temperatura).*

Ahora, para ilustrar un problema inverso consideremos el siguiente ejemplo:

Ejemplo 2.2. *Supongamos que ahora deseamos determinar la temperatura dentro de un horno, debido a que el rango de temperatura es muy elevado, no podemos emplear un termómetro convencional. Para resolver el problema podemos utilizar ultrasonido: en este caso la temperatura altera las propiedades acústicas del aire modificando la forma en que*

¹Tomado de [23]

las ondas de sonido se mueven en el medio. Así, el modelo directo corresponde al análisis de la propagación de las ondas de sonido en el aire en función de la temperatura mientras que el modelo inverso corresponde a determinar la temperatura a partir de la información obtenida de la propagación de las ondas de sonido.

Si denotamos por \mathcal{A} al operador que describe el modelo directo y por \mathcal{A}^{-1} al operador que describe el modelo inverso, definidos de manera que

$$\mathcal{A}(\text{causa}) = \text{efecto}.$$

Entonces, un problema inverso se plantea como

$$\mathcal{A}^{-1}(\text{efecto}) = \text{causa}.$$

En la Figura 2.1 se considera la relación entre el desenfoque de una imagen y la imagen original. En este caso, el problema directo consiste en obtener la imagen desenfocada a partir de la imagen original, mientras que el problema inverso consiste en obtener la imagen original a partir de la imagen desenfocada².

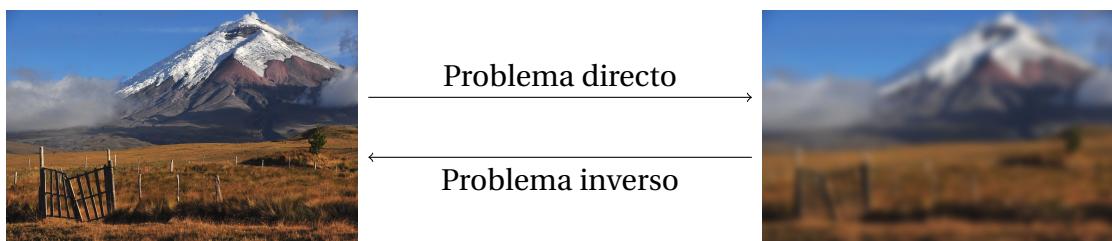


Figura 2.1: Ejemplo de un problema directo e inverso.

2.1.1. Problemas mal condicionados

Una vez definidos los problemas directo e inverso, consideraremos la noción de un *problema bien puesto* dada por Hadamard en [19]. El cumplimiento de los criterios mencionados en la siguiente definición nos brindan una idea de la complejidad o dificultad que podemos encontrarnos al resolver un problema.

²Una de las técnicas más comunes para realizar el desenfoque de una imagen es utilizar la operación convolución con una aplicación adecuada según los objetivos. Así, el obtener la imagen desenfocada corresponde a obtener la *deconvolución*.

Definición 2.1. *Un problema bien puesto es aquel que cumple con las siguientes condiciones:*

- 1. Existencia:** *El problema admite al menos una solución.*
- 2. Unicidad:** *La solución es única.*
- 3. Estabilidad:** *La solución varía de manera continua con respecto a los datos iniciales.*

A partir de la definición anterior se dice que un problema está *mal puesto* si no se cumplen al menos las condiciones de existencia y estabilidad.

En general, para el caso de imágenes médicas el modelo base es el siguiente:

$$\mathcal{A}f + \varepsilon = m, \quad (2.1)$$

donde f es una función continua por partes, m es el vector de datos obtenidos a partir del equipo de imagen y ε es el error proveniente de la medición. Puesto que la implementación de los modelos es discreta y los datos obtenidos son finitos, el problema se plantea considerando a f como un vector y \mathcal{A} una matriz que describe la relación entre f y m . A partir de la ecuación (2.1), intuitivamente, estamos tentados a realizar la siguiente conjectura

$$\mathcal{A}^{-1}m \approx f.$$

No obstante, dado que en general los problemas que modelan imágenes médicas son mal puestos, la solución no es única y en muchos casos el problema es inestable, lo que puede llevar a problemas debido al ruido en los datos obtenidos. En consecuencia, la conjectura anterior no es válida y esto nos lleva al estudio de otras técnicas para la resolución de problemas.

2.1.2. Crímenes inversos

Una vez definido el problema, una de las aproximaciones más utilizada es recurrir a los métodos numéricos para encontrar una solución aproximada. En [23] se menciona que los *crímenes inversos* se resumen en que: «el modelo y la realidad coinciden», es

decir, la solución obtenida por métodos numéricos es la solución exacta del problema³. Adicionalmente, [23] menciona que se puede caer en un crimen inverso en las siguientes situaciones:

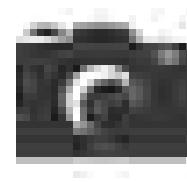
1. Los datos empleados para probar el modelo inverso son generados por el mismo modelo.
2. La discretización utilizada para los datos de entrada del modelo directo es igual a la discretización utilizada para los datos empleados en el modelo inverso.

2.2. Procesamiento básico de imágenes

En esta sección abordaremos los conceptos básicos del procesamiento de imágenes, así como algunas definiciones y herramientas utilizadas para su manejo y procesamiento. Para ilustrar los conceptos mencionados consideremos la imagen de la Figura 2.2a de 432×288 píxeles. Esta será reducida a una imagen de 25×25 píxeles para facilitar la explicación de los conceptos; en la Figura 2.2b se muestra la imagen reducida.



(a) Icono cámara.



(b) Icono cámara reducido a 25×25 píxeles.

Figura 2.2: Icono cámara en diferentes resoluciones.

La Tabla 2.1 muestra los valores de los píxeles de la imagen de la Figura 2.2a, considerando una escala de grises de 0 a 255. Adicionalmente, utilizaremos la imagen de la Figura 2.3 para ilustrar los conceptos que no requieran de una imagen reducida.

Por otro lado, una imagen también puede ser vista como una función en la cual a cada par de coordenadas (x, y) asignamos un valor de intensidad. En este caso, simbólicamente podemos escribir

$$f : \Omega \subseteq \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{R},$$

³En otras áreas como el aprendizaje automático, el término *overfitting* se utiliza para describir una situación similar, en la que el modelo se ajusta de manera excesiva a los datos proporcionados para su entrenamiento pero no es capaz de generalizar a nuevos datos.

| | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 255 | 255 | 255 | 255 | ... | 255 | 255 | 255 | 255 |
| 255 | 255 | 255 | 252 | ... | 255 | 255 | 255 | 255 |
| 255 | 255 | 251 | 243 | ... | 245 | 248 | 255 | 255 |
| 247 | 139 | 82 | 77 | ... | 77 | 77 | 114 | 229 |
| : | : | : | : | .. | : | : | : | : |
| 148 | 77 | 77 | 77 | ... | 77 | 77 | 77 | 90 |
| 166 | 77 | 77 | 77 | ... | 77 | 77 | 77 | 114 |
| 236 | 103 | 77 | 77 | ... | 77 | 77 | 85 | 210 |
| 255 | 251 | 222 | 213 | ... | 213 | 217 | 246 | 255 |

Tabla 2.1: Valores de los píxeles de la imagen de la Figura 2.2a.



Figura 2.3: Imagen original.

donde la imagen está en escala de grises. Para modelar una imagen a color, se consideran tres funciones f_r , f_g y f_b que representan los valores de intensidad de los canales rojo, verde y azul, respectivamente, en ese caso se expresa como

$$\mathbf{f}(x, y) = (f_r(x, y), f_g(x, y), f_b(x, y))$$

para el caso de imágenes en formato RGB.⁴ Así, para simplificar la notación consideraremos imágenes en escala de grises para las secciones posteriores; no obstante, los conceptos presentados son aplicables a imágenes a color con las modificaciones necesarias para considerar los diferentes canales de color.

⁴El formato RGB corresponde a las siglas *Red*, *Green*, *Blue* bajo este modelo se construyen todos los colores mostrados en pantalla como una combinación de los colores rojo, verde y azul.

2.2.1. Imágenes e histogramas

En la Figura 2.4 se muestra un ejemplo para la versión en escala de grises de la Figura 2.3. El histograma es una representación gráfica de la distribución de los valores de intensidad de una imagen. En este caso, el eje x representa los valores de intensidad de 0 a 255 y el eje y representa la cantidad de píxeles con dicha intensidad. Adicionalmente, a partir del histograma se puede obtener la *función de distribución acumulada* (CDF, por sus siglas en inglés) en la Figura 2.4c se puede observar la CDF asociada.

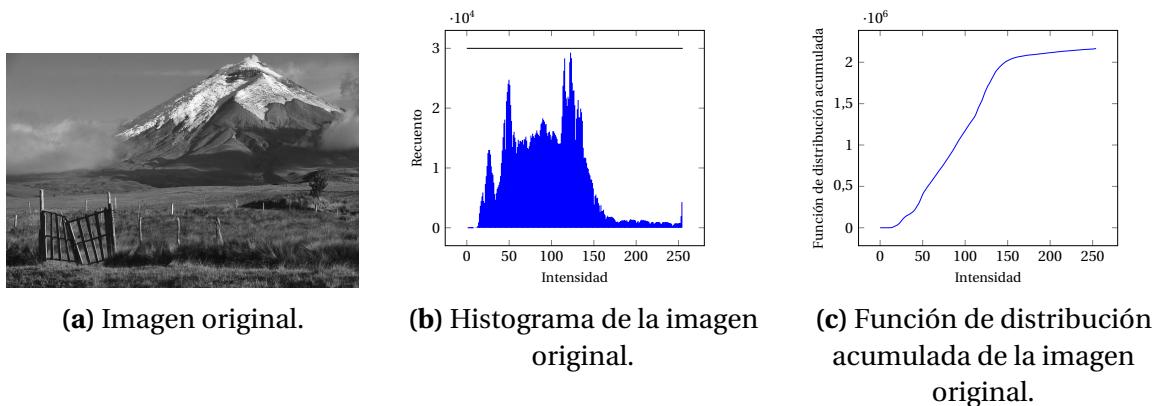


Figura 2.4: Imagen original junto con su histograma y función de distribución acumulada.

2.2.2. Ecualización de histogramas

La ecualización de histogramas es un proceso que permite mejorar la calidad de la imagen. Este proceso se basa en distribuir uniformemente los valores de intensidad de la imagen y, en consecuencia, obtenemos como resultado una función de distribución acumulada lineal. En la Figura 2.5 realizamos la ecualización del histograma para la Figura 2.4.

2.2.3. Detección de bordes

La detección de bordes es uno de los problemas más comunes en el procesamiento de imágenes. En general, el ojo humano percibe los bordes como los cambios bruscos en la intensidad de la imagen, de esta manera, la detección de bordes se puede realizar

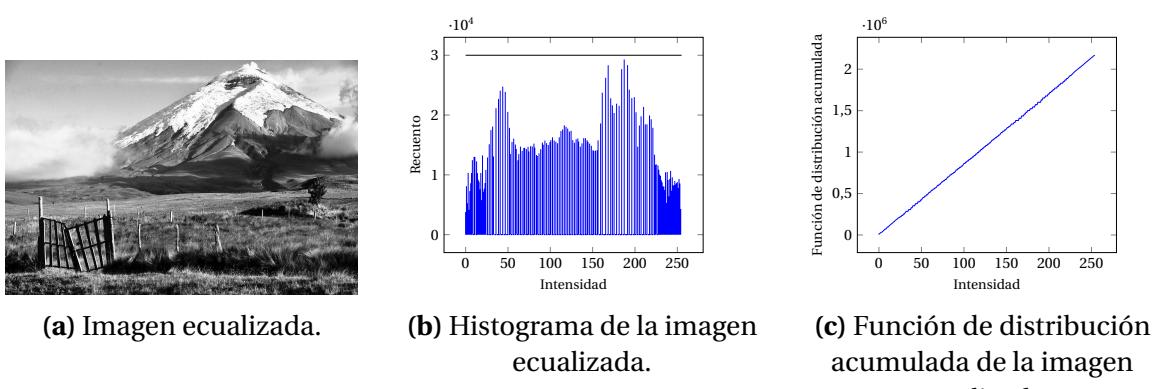


Figura 2.5: Ecualización de histogramas.

a partir del cálculo de la derivada de la imagen. Para este fin, dado que consideramos una imagen bidimensional utilizaremos *diferencias finitas* las cuales se definen para un punto (a, b) , un incremento h y una dirección x o y como

$$\Delta_x f(a, b) = f(a + h, b) - f(a, b), \quad (2.2)$$

$$\Delta_y f(a, b) = f(a, b + h) - f(a, b). \quad (2.3)$$

para x y y , respectivamente. Dado que h es positivo, a las expresiones (2.2) y (2.3) se les conoce como diferencias finitas *hacia adelante*.

En la Figura 2.6 se muestra las derivadas de la imagen de la Figura 2.3 en las direcciones x y y normalizadas para una mejor visualización. Nuevamente, se considera una imagen bidimensional, por lo tanto, en escala de grises.

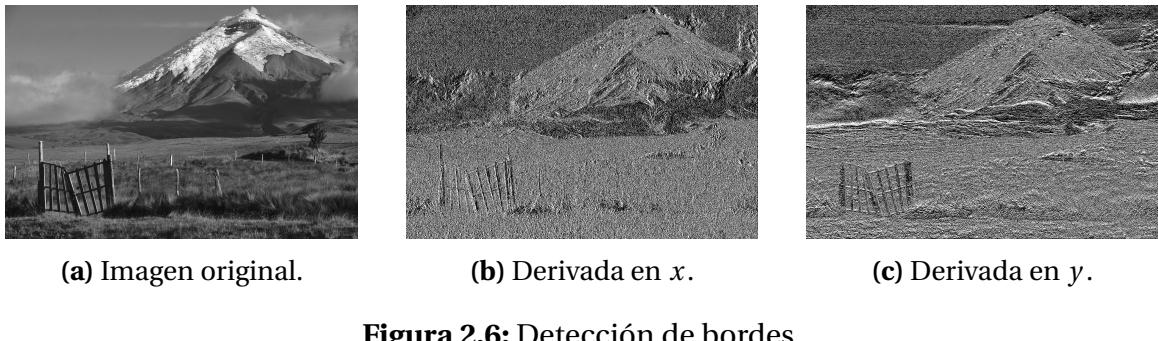


Figura 2.6: Detección de bordes.

De la misma manera, también es posible realizar el cálculo empleando diferencias finitas *hacia atrás* o *centradas* en donde la diferencia entre las diferentes aproximaciones es la aplicabilidad de la derivada en los bordes de la imagen y la precisión de la aproxi-

mación. En las ecuaciones (2.4) y (2.5) se presentan las diferencias finitas hacia atrás y centradas, respectivamente en la dirección x .

$$\nabla_x f(a, b) = f(a, b) - f(a - h, b), \quad (2.4)$$

$$\delta_x f(a, b) = \frac{f(a + h, b) - f(a - h, b)}{2h}. \quad (2.5)$$

Finalmente, como el dominio de las imágenes corresponde a índices enteros, usualmente se considera $h = 1$ en las diferencias finitas.

2.3. Transformada de Fourier

La transformada de Fourier es una herramienta matemática de gran utilidad en el análisis de señales y sistemas para la ingeniería, solución de ecuaciones en derivadas parciales, etc. En esta sección vamos a introducir algunas propiedades y resultados importantes, para una exposición más detallada se recomienda consultar [18].

Definición 2.2 (Transformada de Fourier). *Sea $f \in L^1(\mathbb{R}^n)$, la transformada de Fourier de f se define como*

$$\widehat{f}(\xi) = \int_{\mathbb{R}^n} f(x) e^{-2\pi i x \cdot \xi} dx. \quad (2.6)$$

Además, se define la transformada inversa de Fourier como

$$f(x) = \int_{\mathbb{R}^n} \widehat{f}(\xi) e^{2\pi i x \cdot \xi} d\xi. \quad (2.7)$$

Aún cuando la Definición 2.2 solo considera funciones en $L^1(\mathbb{R}^n)$, es posible extender la definición a otros espacios como $L^2(\mathbb{R}^n)$, funciones a decaimiento rápido $\mathcal{S}(\mathbb{R}^n)$ y distribuciones temperadas $\mathcal{S}'(\mathbb{R}^n)$. La siguiente proposición muestra una propiedad importante de la transformada de Fourier cuya demostración es evidente a partir de la Definición 2.2.

Proposición 2.1. *La Transformada de Fourier como operador es una transformación lineal.*

Observación 2.1. Por facilidad, se denotará a la transformada de Fourier de una función f como $\mathcal{F}(f)$.

Antes de continuar, vamos a introducir algunas notaciones usuales en el análisis de Fourier. Sea $f: \mathbb{R}^n \rightarrow \mathbb{R}$ una función medible, $x \in \mathbb{R}^n$ y $a > 0$, entonces se define:

1. Traslación de f : $(\tau^y f)(x) = f(x - y)$.
2. Dilatación de f : $(\delta^a f)(x) = f(ax)$.
3. Reflexión de f : $(\tilde{f})(x) = f(-x)$.

2.3.1. Espacio de Schwartz

Con el fin de generar un marco funcional apropiado en el que podamos aprovechar todas las buenas propiedades de la transformada de Fourier, vamos a introducir la notación de espacio de Schwartz. Así, como veremos posteriormente, tendremos varias propiedades útiles para el análisis de señales y sistemas.

Definición 2.3 (Espacio de Schwartz). *Se define el espacio de Schwartz, también llamado espacio de funciones a decaimiento rápido o clase de funciones de Schwartz, como el conjunto*

$$\mathcal{S}(\mathbb{R}^n) = \left\{ \phi \in C^\infty(\mathbb{R}^N) : \sup_{x \in \mathbb{R}^n} |x^\beta \partial^\alpha \phi(x)| < +\infty \quad \forall \alpha, \beta \in \mathbb{N}_0^n \right\}$$

Proposición 2.2. Sean $f, g \in \mathcal{S}(\mathbb{R}^n)$, $y \in \mathbb{R}^n$, $b \in \mathbb{C}$, $\alpha \in \mathbb{N}_0^n$ y $t > 0$, entonces se cumplen las siguientes propiedades:

1. $\|\mathcal{F}f\|_\infty \leq \|f\|_{L^1}$,
2. $\mathcal{F}(f+g) = \mathcal{F}f + \mathcal{F}g$,
3. $\mathcal{F}(bf) = b\mathcal{F}f$,
4. $\mathcal{F}\tilde{f} = \widetilde{\mathcal{F}f}$,
5. $\mathcal{F}\bar{f} = \overline{\widetilde{\mathcal{F}f}}$,
6. $\mathcal{F}(\tau^y f)(\xi) = e^{-2\pi i y \cdot \xi} \mathcal{F}f(\xi)$,

7. $\mathcal{F}(\delta^t f)(\xi) = \delta^{t^{-1}} t^{-n} \mathcal{F}f\left(\frac{\xi}{t}\right)(\xi),$
8. $\mathcal{F}(\partial^\alpha f) = (2\pi i \xi)^\alpha \mathcal{F}f,$
9. $(\partial^\alpha \mathcal{F}f)(\xi) = \mathcal{F}((-2\pi i x)^\alpha f)(\xi),$
10. $\mathcal{F}f \in \mathcal{S}(\mathbb{R}^n),$
11. $\mathcal{F}(f * g) = \mathcal{F}f \mathcal{F}g,$
12. $\mathcal{F}(f \circ A)(\xi) = \mathcal{F}(A\xi)$ con A una matriz ortogonal y ξ un vector columna.

Demostración. Proposición 2.2.11, [18]. □

Corolario 2.1. *La transformada de Fourier de una función radial es una función radial. Más aún, productos y convoluciones de funciones radiales son radiales.*

Demostración. Proposición 2.2.12, [18]. □

2.3.2. Transformada de Fourier inversa

En esta sección vamos a introducir la transformada de Fourier inversa, la cual nos permite recuperar la función original a partir de su transformada.

Definición 2.4 (Transformada de Fourier inversa). *Sea $f \in \mathcal{S}(\mathbb{R}^n)$, se define la transformada de Fourier inversa de f como*

$$\mathcal{F}^{-1}(\xi) = \int_{\mathbb{R}^n} f(x) e^{2\pi i x \cdot \xi} dx. \quad (2.8)$$

Observación 2.2. *A partir de la ecuación (2.8) se tiene que*

$$\mathcal{F}^{-1}f(x) = \mathcal{F}f(-x).$$

Dada la definición de la transformada de Fourier inversa, no es difícil ver que esta mantiene propiedades análogas a las enunciadas en la Proposición 2.2. Finalmente, estamos interesados en estudiar la relación entre la transformada de Fourier y la transformada de Fourier inversa, para ello, consideraremos la siguiente proposición.

Teorema 2.1. Sean f, g y $h \in \mathcal{S}(\mathbb{R}^n)$, entonces se cumplen las siguientes propiedades:

1. $\int_{\mathbb{R}^n} f(x) \mathcal{F}(g(x)) dx = \int_{\mathbb{R}^n} \mathcal{F}(f(x)g(x)) dx.$
2. (**Inversión de Fourier**) $\mathcal{F}^{-1}(\mathcal{F}f) = f = \mathcal{F}(\mathcal{F}^{-1}f).$
3. (**Relación de Parseval**) $\int_{\mathbb{R}^n} f(x) \overline{g(x)} dx = \int_{\mathbb{R}^n} \mathcal{F}f(\xi) \overline{\mathcal{F}g(\xi)} d\xi.$
4. (**Identidad de Plancherel**) $\int_{\mathbb{R}^n} f(x) \overline{g(x)} dx = \int_{\mathbb{R}^n} \mathcal{F}f(\xi) \overline{\mathcal{F}g(\xi)} d\xi.$
5. $\int_{\mathbb{R}^n} f(x)h(x) dx = \int_{\mathbb{R}^n} \mathcal{F}f(x) \mathcal{F}^{-1}h(x) dx.$

Demostración. Teorema 2.2.14, [18]. □

Corolario 2.2. La transformada de Fourier es un operador homeomorfo de $\mathcal{S}(\mathbb{R}^n)$ en $\mathcal{S}(\mathbb{R}^n)$.

Demostración. Corolario 2.2.15, [18]. □

2.4. Transformada de Radón

La transformada de Radón nos permite estudiar la relación entre una función y sus proyecciones en un espacio de menor dimensión. En esta sección vamos a introducir la transformada de Radón y algunas de sus propiedades, para una exposición más detallada se recomienda consultar [32, 33].

Puesto que la integración se realiza sobre hiperplanos, conviene introducir la siguiente notación:

$$H(\theta, s) = \{x \in \mathbb{R}^n : x \cdot \theta = s\}, \quad (2.9)$$

con $\theta \in \mathbb{S}^n$ y $s \in \mathbb{R}$, donde \mathbb{S}^n es la esfera unitaria en \mathbb{R}^N . Así, el conjunto H está compuesto por todos los puntos del hiperplano perpendicular a θ y que está a una distancia s del origen.

Observación 2.3. A partir de la ecuación (2.9) es fácil ver que $H(\theta, s) = H(-\theta, -s)$.

Para nuestros propósitos, consideraremos la transformada de Radón de una función $f \in \mathcal{S}(\mathbb{R}^n)$ en \mathbb{R}^n . Esto nos permitirá aprovechar las propiedades de la transformada de Fourier y la transformada de Radón y posteriormente la relación que existe entre estas.

Definición 2.5 (Transformada de Radón). *Sea $f \in \mathcal{S}(\mathbb{R}^n)$, se define la transformada de Radón de f como la función Rf definida en $\mathbb{S}^n \times \mathbb{R}$ definida por*

$$Rf(\theta, s) = \int_{H(\theta, s)} f(x) dx. \quad (2.10)$$

Si ahora consideramos $p(s, \theta) = Rf(\theta, s)$, entonces en la figura 2.7 se muestra la idea geométrica a partir de la cual se define la transformada de Radón y más aún la forma en la que se representan los datos. En efecto, si fijamos θ , entonces obtenemos una función de una variable $p_\theta(s) = Rf(\theta, s)$, la cual se puede denominar como la proyección de f en la dirección θ . En la figura 2.7, dado que el objeto posee un tamaño variable (en las direcciones perpendiculares a θ), la proyección de f genera un aporte distinto de acuerdo a la «cantidad de objeto» que atraviesa el hiperplano $H(\theta, s)$ (el semicírculo observado en el eje del plano). En consecuencia, la transformada nos brinda un plano en el que para cada ángulo podemos conocer la cantidad de objeto que atraviesa el hiperplano representado por la franja coloreada. A este último objeto, se lo conoce como sinograma y es el elemento a partir del cual, en etapas posteriores, se realizará la reconstrucción de la imagen.

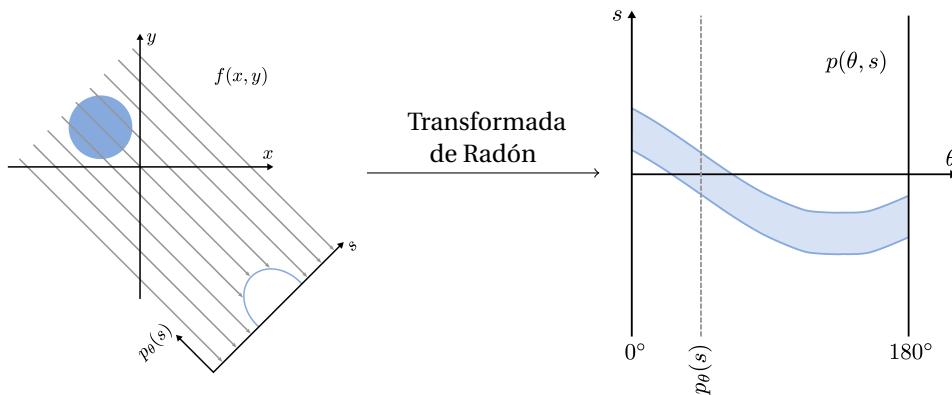


Figura 2.7: Representación de la transformada de Radón. Adaptado de [26].

Gracias a la Observación 2.3 se tiene que $Rf(\theta, s) = Rf(-\theta, -s)$, es decir, la transformada de Radón es una función par. Además, una expresión equivalente a la ecua-

ción (2.10) es

$$Rf(\theta, s) = \int_{\mathbb{R}^n} f(x) \delta(x \cdot \theta - s) dx, \quad (2.11)$$

donde δ es la delta de Dirac en una dimensión. Además, si definimos el cilindro unitario

$$\mathbb{C}^n = \{(\theta, s) : \theta \in \mathbb{S}^n, s \in \mathbb{R}\}, \quad (2.12)$$

entonces la transformada de Radón se puede expresar como una función definida sobre el cilindro unitario y además se tiene que $Rf \in \mathcal{S}(\mathbb{R}^n)(\mathbb{C}^n)$. De esta manera, notemos que muchas de las propiedades de la transformada de Radón se heredan de la transformada de Fourier.

Sobre \mathbb{C}^n se definirá la transformada de Fourier y la convolución sobre la segunda variable, es decir, para $f, g \in \mathcal{S}(\mathbb{R}^n)(\mathbb{C}^n)$ se define:

$$\mathcal{F}(f)(\theta, \xi) = \int_{\mathbb{R}} e^{-2\pi i s \xi} f(\theta, s) ds, \quad (2.13)$$

$$(f * g)(\theta, s) = \int_{\mathbb{R}} f(\theta, s-t) g(\theta, t) dt. \quad (2.14)$$

En consecuencia, tenemos el siguiente resultado sobre la convolución de la transformada de Radón.

Teorema 2.2. *Sean $f, g \in \mathcal{S}(\mathbb{R}^n)$, entonces se cumple que*

$$R(f * g) = Rf * Rg,$$

donde la convolución del lado izquierdo se realiza sobre \mathbb{C}^n y la convolución del lado derecho se realiza sobre \mathbb{R}^n .

Observación 2.4. *Como se mencionó al inicio de esta sección, la transformada de Radón integra una función sobre hiperplanos en \mathbb{R}^n . No obstante, para el caso particular $n = 2$, esta coincide con la transformada de rayos X⁵ en la que se integra sobre líneas rectas. Por lo tanto, para este caso particular, ambas transformadas son equivalentes y difieren solo en notación.*

⁵Ver sección 2.2 [33]

Retroproyección filtrada

A partir del sinograma definido en la sección anterior y notando que las variables del sinograma están definidas en

$$(s, \theta) \in [-R_F, R_F] \times [0, \pi]. \quad (2.15)$$

Con esto, y a partir del modelo definido en (2.37) podemos destacar las siguientes características:

- 1. Invarianza por traslaciones:** La transformación de la imagen trasladada es el desplazamiento de cada sinograma, es decir, $f_t(x, y) = f(x - t_x, y - t_y)$ es $Tf_t(s, \phi) = Tf(s - t_x \cos(\phi) - t_y \sin(\phi), \phi)$.
- 2. Invarianza por rotaciones:** La rotación de la imagen es equivalente a la rotación de cada sinograma, es decir, $f_\theta(x, y) = f(x \cos(\theta) - y \sin(\theta), x \sin(\theta) + y \cos(\theta))$ es $Tf_\theta(s, \phi) = Tf(s, \phi + \theta)$.

Gracias a estas características el algoritmo descrito es válido solo cuando el escáner es capaz de detectar todas las líneas que cruzan el soporte de la imagen (el disco de centro el eje del escáner y radio R_F), es decir, el sinograma es obtenido de todo el espacio definido en (2.15). Cuando esta condición no se satisface se dice que se tiene un *problema de datos incompletos*⁶, no obstante, para la aplicación del algoritmo FBP⁷ se deben corregir los datos estimando las LOR perdidas.

El algoritmo FBP es una de las técnicas clásicas para abordar el problema de la reconstrucción de imágenes en PET. Debido a la relación entre la Transformada de Radón (o la Transformada de Rayos X) y la Transformada (Rápida)⁸ de Fourier el algoritmo se cataloga como un método rápido y eficiente. Así, este se puede resumir en los siguientes pasos:

$$f(x, y) = (T^* p^F)(x, y) = \int_0^\pi p^F(s, \phi) d\phi = \int_0^\pi p^F(x \cos(\phi) + y \sin(\phi), \phi) d\phi, \quad (2.16)$$

⁶Para más información ver [32].

⁷En inglés *Filtered Backprojection*.

⁸En inglés *Fast Fourier Transform*, FFT

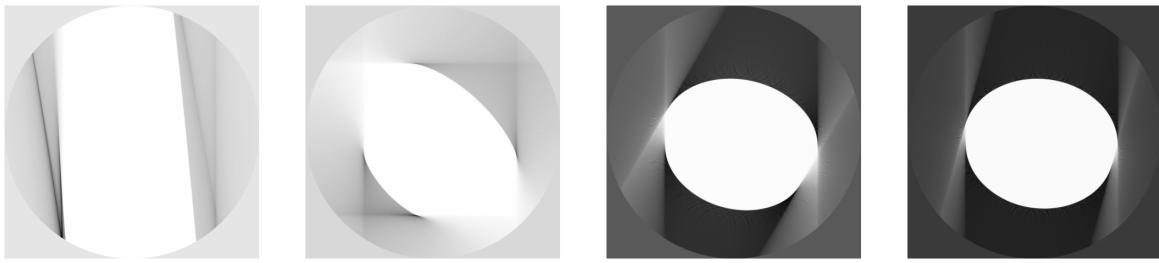
donde

$$p^F(s, \phi) = \int_{-R_F}^{R_F} p(s', \phi) h(s - s') ds', \quad (2.17)$$

y el filtro h es de la forma

$$h(s) = \int_{-\infty}^{\infty} |\nu| e^{2\pi i s \nu} d\nu. \quad (2.18)$$

En la Figura 2.8 se muestra el progreso del algoritmo FBP para la reconstrucción de la Figura 2.20, note que conforme se toman en cuenta más ángulos, es decir, más proyecciones, la imagen reconstruida es más precisa.



(a) 10 proyecciones. (b) 90 proyecciones. (c) 150 proyecciones. (d) 165 proyecciones.

Figura 2.8: Progreso del algoritmo FBP para la reconstrucción de la Figura 2.20.

Observación 2.5. El operador T^* que devuelve f a partir de p se denomina retroproyección y corresponde al operador dual de la transformación de rayos X. Geométricamente, $T^* p^F(x, y)$ es la suma de todos los datos filtrados p^F en todas las líneas que contienen al punto (x, y)

Observación 2.6. La convolución en (2.17) se puede expresar utilizando la Proposición 2.2 como

$$p^F(s, \phi) = \mathcal{F}(p(s, \phi))(\nu, \phi) \cdot \mathcal{F}(h(s))(\nu).$$

2.4.1. Teorema del corte central

En esta sección vamos a abordar la relación entre la transformada de Radón y la transformada de Fourier, la misma que, como veremos más adelante, será fundamental para realizar la reconstrucción de imágenes a partir de sus proyecciones. Para un tratamiento riguroso se recomienda consultar [32, 33] mientras que una aproximación intuitiva se puede encontrar en [26, 14].

Gracias a lo hecho previamente, hemos construido un marco apropiado para el estudio de la proyección de un objeto mediante la transformada de Radón. Luego, como la transformada de Radón admite transformada de Fourier, es posible obtener la transformada de Fourier de las proyecciones de un objeto. El siguiente teorema nos muestra que la transformada de Fourier de las proyecciones de un objeto es igual a la transformada de Fourier del objeto evaluada en una dirección particular.

Teorema 2.3 (Teorema del corte central o del corte de Fourier). *Sea $f \in \mathcal{S}(\mathbb{R}^n)$, entonces se cumple que*

$$\mathcal{F}(Rf)(\theta, \xi) = \mathcal{F}(f)(\theta \xi),$$

donde $\xi \in \mathbb{R}$ y $\theta \in \mathbb{S}^n$.

Demostración. Notando $p_\theta(s) = Rf(\theta, s)$ y gracias a la ecuación (2.13) se tiene que

$$\mathcal{F}(Rf)(\theta, \xi) = \int_{\mathbb{R}} e^{-2\pi i s \xi} p_\theta(s) ds.$$

Reemplazando la definición de la transformada de Radón (ecuación (2.11)) y aplicando el Teorema de Fubini-Tonelli se obtiene

$$\begin{aligned} \int_{\mathbb{R}} e^{-2\pi i s \xi} p_\theta(s) ds &= \int_{\mathbb{R}} e^{-2\pi i s \xi} \int_{\mathbb{R}^n} f(x) \delta(x \cdot \theta - s) dx ds, \\ &= \int_{\mathbb{R}^n} f(x) \int_{\mathbb{R}} e^{-2\pi i s \xi} \delta(x \cdot \theta - s) ds dx, \\ &= \int_{\mathbb{R}^n} f(x) e^{-2\pi i x \cdot \theta \xi} dx. \end{aligned}$$

Así, por la definición de la transformada de Fourier (ecuación (2.13)) se tiene que

$$\mathcal{F}(Rf)(\theta, \xi) = \mathcal{F}(f)(\theta \xi),$$

lo que concluye la demostración. \square

Finalmente, la Figura 2.9 nos muestra la idea a partir de la cual vamos a aplicar la transformada de Radón y la transformada de Fourier para obtener la imagen original. A continuación se describen los pasos a seguir:

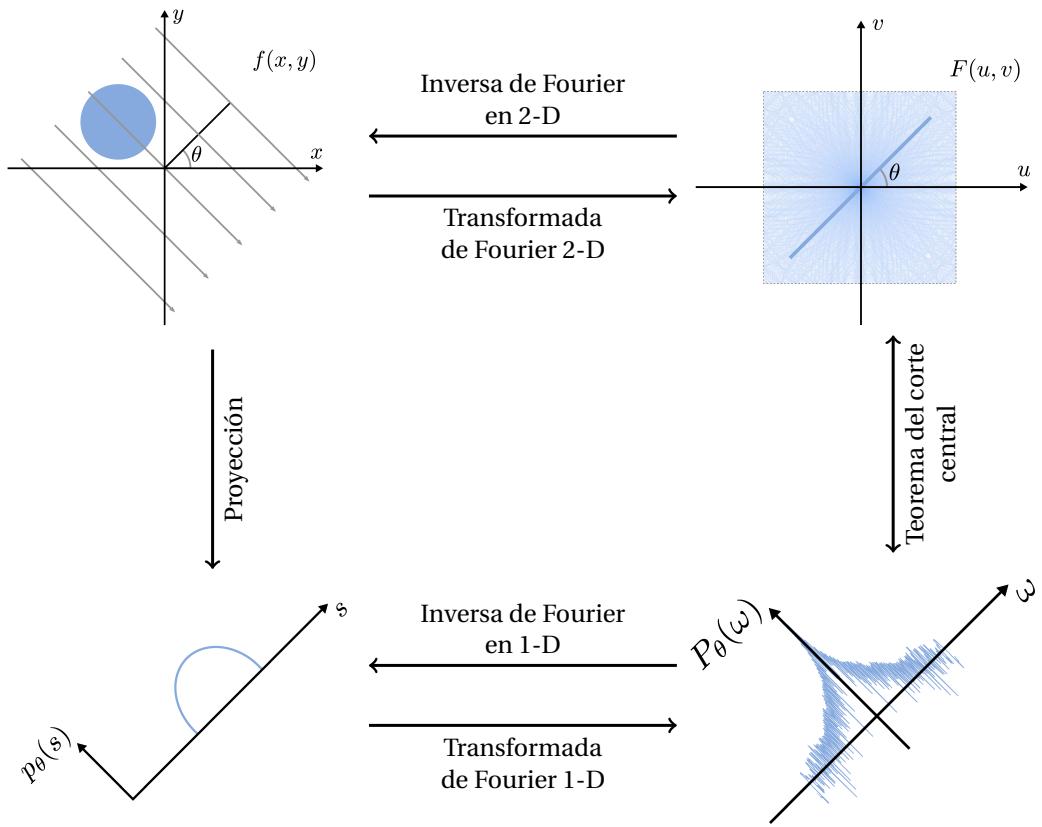


Figura 2.9: Representación de la aplicación del teorema del Teorema 2.3.
Adaptado de [26].

1. Se obtienen las proyecciones del objeto $f(x, y)$ a estudiar mediante la transformada de Radón; obteniendo ahora las proyecciones dependientes de (θ, s) . En la práctica, estas proyecciones se obtienen mediante un tomógrafo (también mediante un escáner PET) el cual registra los datos en un sinograma.
2. Dado que se considera un marco funcional adecuado (espacio de Schwartz), es posible aplicar la transformada de Fourier y su inversa en la variable real (en general esto no es posible). En consecuencia, se obtienen las transformadas de Fourier de las proyecciones.
3. Gracias al teorema del corte central, se obtiene una equivalencia que nos permite «intercambiar» los roles de la transformada de Radón y la transformada de Fourier. Así, de la transformada de Fourier de las proyecciones se deduce que es igual a la transformada de Fourier de la imagen original evaluada en una dirección particular.
4. Finalmente, y como nuevamente consideramos un marco funcional adecuado, se

aplica la transformada inversa de Fourier para obtener la imagen original, $f(x, y)$.

2.5. Nociones básicas de convexidad

En esta sección vamos a presentar algunas nociones básicas de convexidad que serán útiles en la descripción de los métodos de optimización que se presentarán en las siguientes secciones, para profundizar en el tema se recomienda consultar [11, 7].

Denotaremos por \mathcal{X} y \mathcal{Y} dos espacios de Hilbert o Euclídeos provistos de la norma inducida por el producto interno, es decir, $\|\cdot\| = \langle \cdot, \cdot \rangle^{1/2}$.

Definición 2.6 (Función convexa). *Sea $f: \mathcal{X} \rightarrow [-\infty, \infty]$ una función en los reales extendidos. Se dice que f es convexa si y solo si su epígrafe:*

$$\text{epi}(f) = \{(x, t) \in \mathcal{X} \times \mathbb{R} \mid f(x) \leq t\},$$

es un conjunto convexo, es decir, si para todo $\lambda \geq f(x)$, $\mu \geq f(y)$ y $t \in [0, 1]$, se tiene que

$$t\lambda + (1-t)\mu \geq f(tx + (1-t)y).$$

Observación 2.7.

1. *Note que la definición de convexidad presentada nos permite evitar el caso $(+\infty) + (-\infty)$.*
2. *En el caso en el que f no sea idénticamente $+\infty$ y en ningún punto $-\infty$, se recupera la definición usual de convexidad.*
3. *Se dice que f es propia si no es idénticamente $+\infty$ y en ningún punto $-\infty$.*

Definición 2.7 (Función semi-continua inferiormente). *Sea $f: \mathcal{X} \rightarrow [-\infty, \infty]$ una función en los reales extendidos. Se dice que f es semi-continua inferiormente si para todo $\lambda \in \mathbb{R}$ el conjunto*

$$[f \leq \lambda] = \{x \in \mathcal{X} : f(x) \leq \lambda\},$$

es cerrado.

Observación 2.8. Una caracterización para la definición de semi-continuidad inferior es que para todo $x \in \mathcal{X}$ tal que $x_n \rightarrow x$ se tiene que

$$\liminf_{n \rightarrow \infty} f(x_n) \geq f(x).$$

Ejemplo 2.3 (Función indicadora). Dado un conjunto $K \subseteq \mathcal{X}$, la función indicadora de K se define como

$$\delta_K(x) = \begin{cases} 0 & \text{si } x \in K; \\ +\infty & \text{caso contrario.} \end{cases}$$

Note que δ_K es una función convexa si y solo si K es un conjunto convexo y semi-continua inferiormente si y solo si K es cerrado.

Definición 2.8 (Función conjugada). Sea $f: \mathcal{X} \rightarrow [-\infty, +\infty]$ una función en los reales extendidos. La función conjugada de f , como

$$f^*(y) = \sup_{x \in \mathcal{X}^*} (\langle x, y \rangle - f(x)) \quad \forall y \in \mathcal{X}.$$

Observación 2.9. Note que f^* es una función convexa y semi-continua inferiormente.

Note que, si iteramos sobre la función conjugada de una función convexa, obtenemos la siguiente expresión:

$$f^{**}(x) = \sup_{y \in \mathcal{X}^{**}} (\langle y, x \rangle - f^*(y)) \quad \forall x \in \mathcal{X}. \quad (2.19)$$

Además, el siguiente resultado nos da una relación entre la función y su bi-conjugada. Una demostración detallada de este resultado se puede encontrar en [7].

Teorema 2.4 (Teorema de Fenchel-Moreau). Sea $f: \mathcal{X} \rightarrow [-\infty, +\infty]$ una función convexa y semi-continua inferiormente. Entonces, se tiene que

$$f = f^{**}.$$

Definición 2.9 (Subgradiente). Sea $f: \mathcal{X} \rightarrow [-\infty, +\infty]$ una función convexa y semi-continua inferiormente. Se define el Subgradiente de f en $x \in \mathcal{X}$ como

$$\partial f(x) = \{p \in \mathcal{X} : f(y) \geq f(x) + \langle p, y - x \rangle, \quad \forall y \in \mathcal{X}\}.$$

Definición 2.10 (Función proximal). *Sea $f: \mathcal{X} \rightarrow [-\infty, +\infty]$ una función propia, convexa y semi-continua inferiormente, el operador proximal de f se define como*

$$\text{prox}_{\tau f}(x) = \arg \min_{y \in \mathcal{X}} \left(f(y) + \frac{1}{2\tau} \|x - y\|^2 \right),$$

con $\tau > 0$.

Observación 2.10. Una caracterización a través del Subgradiente nos dice que

$$\text{prox}_{\tau f}(x) = p \iff \frac{x - p}{\tau} \in \partial f(p).$$

Ejemplo 2.4 (Proximal de la función indicadora). *En el Ejemplo 2.3 se definió la función indicadora de un conjunto $K \subseteq \mathcal{X}$. Así, si se satisfacen las condiciones de la definición de función proximal, entonces se tiene que*

$$\text{prox}_{\tau \delta_K}(x) = \arg \min_{y \in \mathcal{X}} \left(\delta_K(y) + \frac{1}{2\tau} \|x - y\|^2 \right) = \arg \min_{y \in K} \frac{1}{2\tau} \|x - y\|^2,$$

en particular, para $\tau = 1$ se tiene que lo anterior define la proyección ortogonal de x sobre K .

2.5.1. Métodos de punto de silla - Algoritmos tipo Primal-Dual

En esta sección realizaremos una breve descripción a la teoría de optimización para encontrar puntos de silla, los cuales son especialmente útiles en los problemas relacionados con el tratamiento de imágenes. De manera general, este tipo de métodos se basan en dividir el problema de optimización en sub-problemas más sencillos de abordar que simplifican la búsqueda de soluciones.

Para esta sección, vamos a considerar los problemas en su forma estándar

$$\min_{x \in \mathcal{X}} f(Kx) + g(x), \quad (2.20)$$

donde f y g son funciones convexas «simples»⁹ y $K: \mathcal{X} \rightarrow \mathcal{Y}$ es un operador lineal y acotado. El objetivo es escribir el problema (2.20) como un problema de punto de silla

⁹Decimos que una función convexa es simple si la función proximal τf , con $\tau > 0$ puede ser fácilmente evaluada. Ver [11] Pág 182.

entonces se tiene la siguiente forma equivalente

$$\max_y \inf_x \langle y, Kx \rangle - f^*(y) + g(x). \quad (2.21)$$

Con lo cual, se alterna un descenso proximal en la variable x y una ascenso en la variable dual y , dados de la siguiente forma

$$\begin{aligned} x^{k+1} &= \operatorname{prox}_{\tau g}(x^k - \tau K^* y^k), \\ y^{k+1} &= \operatorname{prox}_{\sigma f^*}(y^k + \sigma K x^{k+1}). \end{aligned}$$

En general, no es claro si las iteraciones con la aproximación propuesta convergen a un punto de silla. No obstante, esta aproximación fue propuesta en [40] para el problema (2.29) presentando resultados satisfactorios especialmente si se combina con un esquema de aceleración. Este esquema se basa en considerar τ decreciente y σ creciente en cada iteración. Más aún, una variante que mejora las características del modelo sin requerir cálculos adicionales fue introducida por [13] y se encuentra detallada en el Algoritmo 1.

Algoritmo 1: Primal-Dual Hybrid Gradient.

Datos: Par de puntos iniciales primal y dual (x^0, y^0) , parámetros $\tau > 0, \sigma > 0$.

para $k \geq 0$ **hacer**

$$\left\{ \begin{array}{l} x^{k+1} = \operatorname{prox}_{\tau g}(x^k - \tau K^* y^k) \\ y^{k+1} = \operatorname{prox}_{\sigma f^*}(y^k + \sigma K x^{k+1}). \end{array} \right. \quad (2.22)$$

fin

Donde el paso de sobre-relajación $2x^{k+1} - x^k = x^{k+1} + (x^{k+1} - x^k)$ se puede interpretar como un extra-gradiante aproximado. Más aún, podemos extender lo realizado previamente a un problema de la forma

$$\min_{x \in \mathcal{X}} f(Kx) + g(x) + h(x), \quad (2.23)$$

donde f, g son convexas, semi-continuas inferiormente y simples, K es un operador lineal acotado y h es convexa con gradiente L_h -Lipschitz. La idea es simplemente realizar un reemplazo del paso de de descenso en x por un paso Forward-Backward con lo cual,

se obtiene

$$x^{k+1} = \text{prox}_{\tau g}(x^k - \tau \nabla h(x^k)). \quad (2.24)$$

Así, el Algoritmo 2 presenta la forma general de la iteración para el algoritmo *Primal-Dual Hybrid Gradient* para este tipo de problemas.

Algoritmo 2: Forma general de la iteración Primal-Dual.

Datos: Puntos $(\bar{x}, \bar{y}, \tilde{x}, \tilde{y})$ y pasos $\tau, \sigma > 0$.

Salida: Puntos $(\tilde{x}, \tilde{y}) = \mathcal{PD}_{\tau, \sigma}(\bar{x}, \bar{y}, \tilde{x}, \tilde{y})$, dados por

$$\begin{cases} \tilde{x} = \text{prox}_{\tau g}(\bar{x} - \tau (\nabla h(\bar{x}) + K^* \tilde{y})), \\ \tilde{y} = \text{prox}_{\sigma f^*}(\bar{y} + \sigma K \tilde{x}). \end{cases} \quad (2.25)$$

Con lo cual, notemos que la ecuación (2.22) corresponde a las iteraciones con pasos fijos τ, σ y $\bar{x} = x^k$, $\bar{y} = \tilde{y} = y^k$ y $\tilde{x} = 2x^{k+1} - x^k$. Para más detalles acerca de la convergencia, aceleración, pre-condicionamiento, etc. ver [11].

Ejemplo 2.5 (Eliminación de ruido estándar utilizando PDHG). *Se deriva del modelo (2.29) definido para la eliminación de ruido estándar en una imagen digital. Primero, utilizando la dualidad, obtenemos el siguiente problema en la forma de punto de silla*

$$\min_u \max_p \langle Du, p \rangle + \frac{1}{2} \|u - u^\diamond\|_2^2 - \delta_{\{\|\cdot\|_{2,\infty} \leq \lambda\}}(p). \quad (2.26)$$

Donde resta detallar los operadores proximales para las funciones

$$g(u) = \frac{1}{2} \|u - u^\diamond\|^2 \quad y \quad f^*(p) = \delta_{\{\|\cdot\|_{2,\infty} \leq \lambda\}}(p).$$

Así, para la función g se tiene que el operador proximal es

$$\hat{u} = \text{prox}_{\tau g}(u) \iff \hat{u}_{i,j} = \frac{\tilde{u}_{i,j} + \tau u_{i,j}^\diamond}{1 + \tau},$$

mientras que la función proximal para $f^*(p)$ está dado por la proyección pixel ortogonal sobre las bolas ℓ_2 de radio λ . (Ver (4.23) [11])

2.6. Introducción a la optimización continua para imágenes

En esta sección vamos a presentar algunos resultados de optimización continua útiles para el trabajo con imágenes. En la primera sección se discutirán tres casos sobre la restauración de imágenes en los cuales modelaremos nuestra situación como un problema de optimización con regularización.

2.6.1. Métricas

Antes de proceder con la descripción de los problemas de optimización, es necesario disponer de herramientas que nos permitan determinar la calidad de las imágenes obtenidas. Usualmente el error cuadrático medio (RMS) es una de las primeras opciones para determinar la «diferencia» entre dos vectores, en particular imágenes. No obstante, este valor no es suficiente para determinar la calidad de una imagen obtenida respecto a una imagen de referencia. Por esta razón, se han desarrollado otras funciones que denominaremos métricas¹⁰ que nos permiten realizar una evaluación más precisa tomando en cuenta otros factores como valores estadísticos o relaciones de ruido en las imágenes.

En este sentido, introduciremos dos métricas que utilizaremos a lo largo de este trabajo para evaluar los resultados obtenidos en cada experimento realizado. En primer lugar, presentamos el *Índice de Similitud Estructural*¹¹ que nos permite considerar aspectos como la luminosidad y el contraste de las imágenes. Para una descripción más detallada de la métrica SSIM se recomienda consultar [39] mientras que para la implementación de esta métrica en Python se puede consultar la librería `scikit-image`¹².

Definición 2.11 (*Índice de similitud estructural*). *Sean $n, m \in \mathbb{N}$ y $I, J \in \mathbb{R}^{m \times n}$ dos imágenes, se define el índice de similitud estructural como*

$$SSIM(I, J) = \frac{(2\mu_I\mu_J + C_1)(2\sigma_{IJ} + C_2)}{(\mu_I^2 + \mu_J^2 + C_1)(\sigma_I^2 + \sigma_J^2 + C_2)},$$

¹⁰No confundir con la definición de métrica en el sentido topológico.

¹¹En inglés: Structural similarity index measure (SSIM)

¹²[Documentación: Structural similarity](#)

donde:

1. μ_I y μ_J son las medias de las imágenes I y J respectivamente.
2. σ_I^2 y σ_J^2 son las varianzas de las imágenes I y J respectivamente.
3. σ_{IJ} es la covarianza entre las imágenes I y J .
4. C_1 y C_2 son constantes que se utilizan para estabilizar la división en caso de que el denominador sea muy pequeño.

Ahora, presentamos la *Razón Señal a Ruido Pico*¹³, esta métrica nos permite obtener una razón que expresa la relación entre la señal y el ruido en una imagen. Puesto que es posible expresar esta métrica en términos del error cuadrático medio, introduciremos esta definición primero y posteriormente presentaremos la definición de PSNR.

Definición 2.12 (Error cuadrático medio). *Sean $n, m \in \mathbb{N}$ y $I, J \in \mathbb{R}^{m \times n}$ dos imágenes, definimos el Error Cuadrático Medio entre las imágenes I y J como*

$$MSE(I, J) = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n (I_{i,j} - J_{i,j})^2.$$

Definición 2.13 (Razón señal a ruido pico). *Sean $n, m \in \mathbb{N}$ y $I, J \in \mathbb{R}^{m \times n}$ dos imágenes, definimos la Razón Señal a Ruido Pico entre las imágenes I y J como*

$$PSNR(I, J) = 10 \log_{10} \left(\frac{\text{MAX}_I^2}{MSE(I, J)} \right),$$

donde MAX_I es el valor máximo que puede tomar la imagen I .

Observación 2.11. *De la forma en que se definió la métrica PSNR, se considera que la imagen I es la imagen de referencia y la imagen J es la imagen que se desea comparar; así el invertir los roles de las imágenes puede alterar el valor de PSNR.*

Observación 2.12. *Note que la métrica PSNR se expresa en decibeles (dB) y que mientras mayor sea el valor de PSNR mejor es la calidad de la imagen pues la relación entre el ruido presente en la imagen y la señal es mayor.*

¹³En inglés: Peak Signal Noise Ratio (PSNR)

Finalmente, para la implementación de la métrica PSNR en Python se puede consultar la librería `scikit-image`¹⁴.

2.6.2. Restauración de imágenes

Eliminación de ruido

En este caso, consideramos la presencia de ruido Gaussiano en la imagen según el modelo ROF (Rudin-Osher-Fatemi) [36] inspirado en la descripción realizada en [11]. Consideremos una imagen digital escalar valuada $f \in \mathbb{R}^{m \times n}$, donde m y n son las dimensiones de la imagen.¹⁵ De esta manera, introducimos el operador gradiente discreto $D: \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m \times n \times 2}$ definido por

$$\begin{aligned}(Df)_{i,j,1} &= \begin{cases} f_{i+1,j} - f_{i,j} & \text{si } 1 \leq i < m; \\ 0, & \text{caso contrario,} \end{cases} \\ (Df)_{i,j,2} &= \begin{cases} f_{i,j+1} - f_{i,j} & \text{si } 1 \leq j < n; \\ 0, & \text{caso contrario.} \end{cases}\end{aligned}\tag{2.27}$$

Observación 2.13. *En algunos casos es necesario conocer la norma del operador D ; en este caso y según [9] esta se puede estimar por*

$$\|D\| \leq \sqrt{8}.\tag{2.28}$$

De esta manera, el modelo ROF discreto se define por

$$\min_f \left(\lambda \|\mathbb{D}f\|_{p,1} + \frac{1}{2} \|f - f^\diamond\|_2^2 \right),\tag{2.29}$$

donde f^\diamond es la imagen observada con ruido y la variación total discreta se define por

$$\|\mathbb{D}f\|_{p,1} = \sum_{i,j=1}^{m,n} |(Df)_{i,j}|_p = \sum_{i,j=1}^{m,n} \left((Df)_{i,j,1}^p + (Df)_{i,j,2}^p \right)^{1/p},\tag{2.30}$$

¹⁴Documentación: Peak Signal Noise Ratio

¹⁵También se pueden considerar imágenes o señales de dimensión superior; por ejemplo, imágenes en color en tres dimensiones.

es decir, la norma ℓ_1 de la norma p de los gradientes de los píxeles de la imagen. Más aún, el parámetro p se elige de acuerdo a la regularización deseada; por ejemplo, si $p = 1$ se obtiene una regularización de tipo anisotrópico que ofrece una mejor preservación de detalles en ciertas direcciones. Por otro lado, si $p = 2$ se obtiene una regularización isotrópica que ofrece una mejor preservación de detalles en todas las direcciones.

Por otro lado, el parámetro $\lambda > 0$ nos permite controlar el balance entre la fidelidad a los datos y la regularización pues actúa como un peso entre ambos términos de la función objetivo. De esta manera, un valor de λ grande favorece la regularización, es decir la eliminación de ruido y un valor pequeño favorece la fidelidad a los datos, es decir la preservación de los detalles de la imagen.

Para ilustrar los conceptos presentados en esta sección haremos uso de las librerías `scipy`, `numpy` y `skimage` en Python. Así, en la Figura 2.10 se muestra un ejemplo de la aplicación del modelo ROF para la eliminación de ruido en una imagen digital siguiendo la teoría descrita en esta sección y usando el código A.6. En este caso se utilizó una imagen de 512×512 píxeles y la función `random_noise` junto con una varianza de 0,01 para generar la imagen mostrada en la Figura 2.10b. Mientras que para la eliminación del ruido se utilizó la función `denoise_tv_chambolle` con un valor de $\lambda = 10$. No obstante, note que el modelo utilizado por la implementación está dado de la siguiente manera:

$$\min_u \sum_{i=1}^{N-1} \left(|\nabla u_i| + \frac{\lambda}{2} (f_i - u_i)^2 \right),$$

donde f es la imagen con ruido y u es la imagen restaurada, λ es el parámetro de regularización y ∇ es el operador gradiente.

Es importante resaltar que el modelo y la solución implementados en su ejecución no representan una carga computacional significativa. En la Tabla 2.2 se muestra una comparativa entre los tiempos de ejecución promedio en cada una de las instancias de la experimentación.

| Instancia | Tiempo de ejecución promedio (s) |
|-----------|----------------------------------|
| CP1 | 0,5 |
| CP2 | 0,3 |

Tabla 2.2: Tiempo de ejecución promedio en la eliminación de ruido en una imagen digital.



Figura 2.10: Ejemplo de eliminación de ruido en una imagen digital usando el modelo ROF.

Restauración de imágenes borrosas

En este caso, para el modelo general, consideramos la aplicación de un operador lineal A en el término de ajuste del modelo (2.29), es decir

$$\min_f \left(\lambda \|\mathbb{D}f\|_{p,1} + \frac{1}{2} \|Af - f^\circ\|_2^2 \right), \quad (2.31)$$

donde $A: \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{k \times l}$ es lineal, f° es la imagen observada y k y l dependen de la aplicación particular a considerar.

Para nuestro ejemplo, consideramos la restauración de una imagen convolucionada con un filtro de tipo Gaussiano. En este caso, el operador A modela el núcleo del filtro Gaussiano. Es importante notar que el término de ajuste cuadrático de los datos es útil para el caso en el que el ruido Gaussiano tiene media cero y varianza conocida. Para otros tipos de ruido es necesario considerar términos de ajuste apropiados.

Así, nuestro objetivo será minimizar la función de costo (2.31) con $Af = a * f$ donde a es el núcleo del filtro Gaussiano de dimension $k \times l$ y f es la imagen a restaurar. De esta manera, en la Figura 2.11 se muestra un ejemplo de la aplicación del modelo ROF para la restauración de una imagen digital borrosa siguiendo la teoría descrita en esta sección y usando el código A.7. En el mismo, utilizamos una imagen de 512×512 píxeles y un filtro de 25×25 píxeles creado a partir de una matriz con todas las entradas iguales a $1/(5 \cdot 5)$, de modo que al realizar la convolución con la imagen se obtiene un efecto de suavizado. Por otro lado, para la eliminación del ruido se definió el modelo descrito

en (2.31) con un valor de $\lambda = 1 \times 10^{-4}$.



(a) Imagen original.



(b) Imagen borrosa.
SSIM: 0,428, PSNR: 14,345.



(c) Imagen restaurada.
SSIM: 0,669, PSNR: 23,249.

Figura 2.11: Ejemplo de eliminación de ruido en una imagen digital usando el modelo ROF.

Para concluir en la Tabla 2.3 se muestra una comparativa entre los tiempos de ejecución promedio en cada una de las instancias de la experimentación. Para este caso es interesante notar la diferencia en los tiempos de ejecución promedio entre las dos instancias, dado que en CP2 se utilizó una GPU, para la ejecución del algoritmo de optimización lo que lo implica una reducción significativa en el tiempo de ejecución y una mayor facilidad para obtener el resultado buscado.

| Instancia | Tiempo de ejecución promedio (s) |
|-----------|----------------------------------|
| CP1 | 600 |
| CP2 | 0,8 |

Tabla 2.3: Tiempo de ejecución promedio en la eliminación de ruido en una imagen digital.

Restauración de imágenes con ruido sal y pimienta

En base al caso estudiado en la Sección 2.6.2 si ahora suponemos que el ruido es de tipo impulsivo o contiene una cantidad significativa de valores atípicos, es necesario considerar una variación al modelo (2.31). En este caso, se reemplaza el término cuadrático de ajuste con un término dependiente del dato en ℓ_1 . De esta manera, el modelo denominado $TV - \ell_1$ es

$$\min_f (\lambda \|\mathbb{D}f\|_{p,1} + \|f - f^\circ\|_1). \quad (2.32)$$

En la Figura 2.12 se muestra un ejemplo de la aplicación del modelo $TV - \ell_1$ para la restauración de una imagen digital con ruido de tipo sal y pimienta siguiendo la teoría descrita en esta sección y usando el código A.8. En esta utilizamos la función `spnoise` del módulo `scico` con un parámetro de 0,25 es decir, el 25 % de los píxeles de la imagen se ven afectados por el ruido. Mientras que para la eliminación del ruido se definió el modelo descrito en (2.32) con un valor de $\lambda = 6$.



Figura 2.12: Ejemplo de eliminación de ruido en una imagen digital con ruido de tipo sal y pimienta usando el modelo $TV - \ell_1$.

En la Figura 2.13 se muestra un detalle de la imagen restaurada en la Figura 2.12c para ilustrar la calidad de la restauración. En este caso, se observa que la imagen restaurada es capaz de recuperar los detalles de la imagen original y eliminar el ruido de tipo sal y pimienta.

La Tabla 2.4 muestra una comparación entre los tiempos de ejecución promedio en cada una de las instancias de la experimentación. Nuevamente, como en la Tabla 2.3 se observa una reducción en el tiempo de ejecución promedio gracias al uso de una GPU en la instancia CP2.

| Instancia | Tiempo de ejecución promedio (s) |
|-----------|----------------------------------|
| CP1 | 15 |
| CP2 | 0,8 |

Tabla 2.4: Tiempo de ejecución promedio en la eliminación de ruido en una imagen digital.

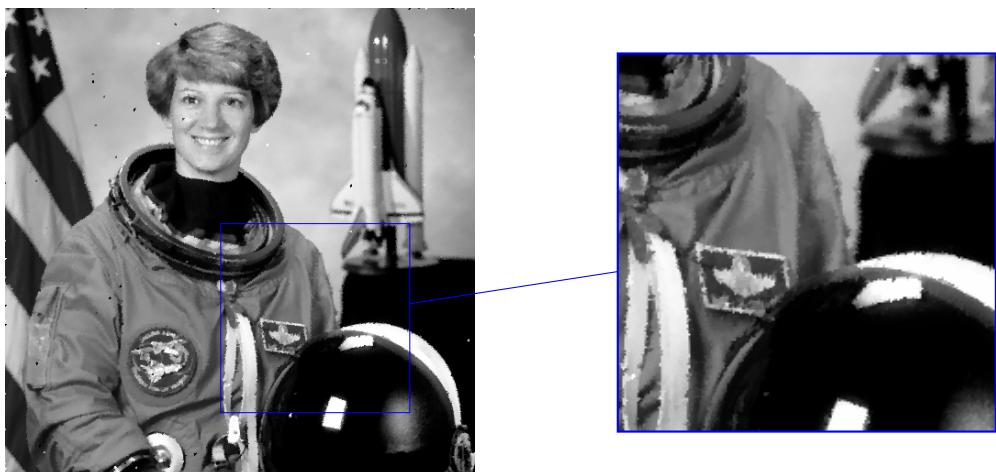


Figura 2.13: Detalle de la imagen restaurada en la Figura 2.12c.

2.7. Tomografía por Emisión de Positrones

2.7.1. Breve historia de las imágenes médicas

La medicina es una de las áreas más desarrolladas por el ser humano, la necesidad de curar o prevenir enfermedades ha llevado al estudio de diferentes técnicas que permitan mejorar la calidad de vida de las personas. En este sentido, aún cuando los exámenes de laboratorio son una herramienta fundamental para el diagnóstico de enfermedades, la medicina nuclear ha permitido el desarrollo de técnicas de diagnóstico y tratamiento de enfermedades de manera no invasiva. El primer paso en este camino fue dado por Wilhelm Conrad Röntgen en 1895 con el descubrimiento de los rayos X, esto permitió por primera vez la visualización del esqueleto sin la necesidad de realizar una cirugía. En la Figura 2.14 se muestra una imagen de rayos X de la mano de la esposa de Röntgen.

A partir de este punto y a lo largo de los años se han desarrollado nuevas y mejores tecnologías que han diversificado los métodos empleados. En este sentido tenemos las imágenes por ultrasonido, útiles para el estudio de tejidos blandos, tomografía computarizada¹⁶ para la visualización de estructuras internas del cuerpo, resonancia magnética nuclear¹⁷ para la visualización de tejidos blandos y la tomografía por emisión de

¹⁶En inglés *Computerized Tomography* (CT).

¹⁷En inglés *Magnetic Resonance Imaging* (MRI).



Figura 2.14: Primera imagen de rayos X de la mano de la esposa de Röntgen.

positrones¹⁸ para el estudio de la actividad metabólica del cuerpo. Una revisión sobre la reconstrucción de imágenes médicas con MR y PET se puede encontrar en [17]. En la Figura 2.15 se muestra una imagen de tomografía computarizada de un cráneo, aquí se puede apreciar con gran detalle las estructuras internas del cráneo.

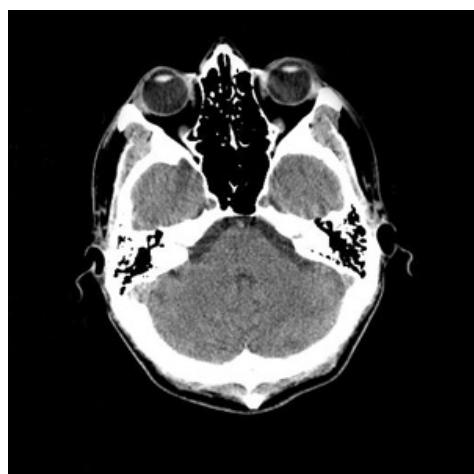


Figura 2.15: Imagen de tomografía computarizada de un cráneo.

2.7.2. Introducción a la tomografía por emisión de positrones

La tomografía por emisión de positrones (PET) es una técnica de imagen médica cuyo objetivo es la visualización de los procesos biológicos que ocurren en el cuerpo, debido a esto se dice que PET es una técnica de *imagen funcional* en contraposición a

¹⁸En inglés *Positron Emission Tomography* (PET).

la tomografía computarizada que es una técnica de *imagen anatómica*. Para lograr su objetivo, PET emplea compuestos radioactivos también denominados *trazadores* que son inyectados en el cuerpo del paciente, estos compuestos poseen dos características fundamentales:¹⁹

1. Los trazadores están diseñados de manera que se acumulen en una región específica del cuerpo, según el interés del estudio.
2. Los trazadores están compuestos por isótopos inestables que emiten radiación durante su transición a un estado estable.

La desintegración radioactiva de un elemento se puede dar, de manera general, de tres maneras:

1. **Radiación α :** consiste en la emisión de un núcleo de helio con una capacidad de penetración pequeña debido a que interacciona con la materia a su alrededor, suele ser detenida por la piel.
2. **Radiación β :** consiste en la emisión de un electrón o un positrón con una capacidad de penetración mayor que la radiación α , requiere delgadas láminas de plomo o aluminio para ser detenida.
3. **Radiación γ :** consiste en la emisión de fotones de alta energía, esta radiación es la más penetrante y requiere de plomo o concreto para diminuir su intensidad.

Para las imágenes funcionales se puede emplear radiación β y γ . Para el primer caso, se utilizan isótopos que emiten positrones, una vez emitidos, estos viajan una distancia muy corta y se aniquilan con un electrón²⁰, este suceso genera un par de fotones con una energía de 511 keV que se desplazan en direcciones diametralmente opuestas²¹ y que son detectadas por el equipo de imagen. En el segundo caso, se emplean isótopos que emiten fotones γ que son detectados por el equipo de imagen denominado *cámara gamma*. En la Figura 2.16a se muestra una representación simplificada de la emisión de fotones para la radiación γ , de la misma manera, en la Figura 2.16b se muestra una representación simplificada de la emisión de fotones para la radiación β .

¹⁹Información detallada sobre los trazadores empleados en PET se puede encontrar en [2].

²⁰A este proceso se lo suele denominar *aniquilación positrón-electrón*

²¹En la práctica, muchos partículas de fotones no se emiten a 180° debido a que al momento de la aniqui-

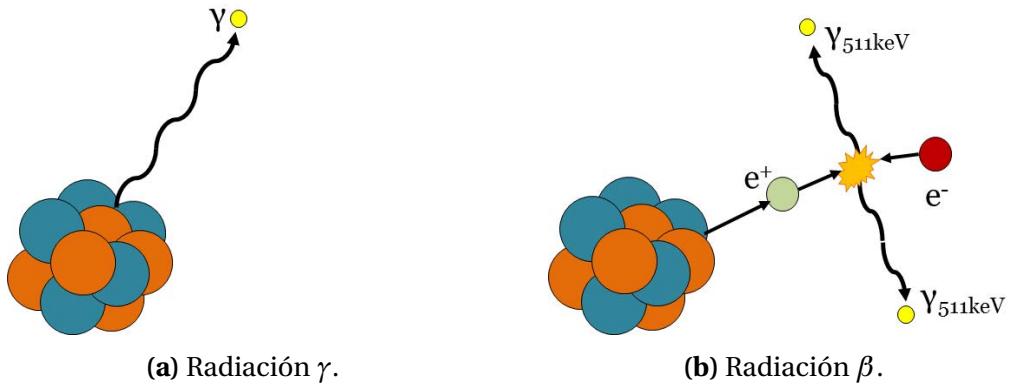


Figura 2.16: Ejemplo de emisión de fotones. Imagen tomada de [26] Pág. 209.

Dada la dependencia de un fenómeno físico, la imagen obtenida por PET es una imagen de *distribución de actividad* que muestra la concentración del trazador. De esta manera, dado que la desintegración radioactiva sigue una distribución de Poisson, la imagen obtenida se puede modelar como una variable aleatoria con la siguiente relación:

$$D \sim \text{Poisson}(A \cdot f), \quad (2.33)$$

donde D representa las mediciones recibidas en el detector, es decir el recuento de fotones, $A \in \mathbb{R}^{M \times N}$ es la *matriz del sistema* en la cual, cada elemento $a_{n,m}$ representa la probabilidad de que un fotón emitido en un vóxel²² n sea contabilizado en el detector m con M el número de detectores y N el número de vóxeles en la imagen, y $f \in \mathbb{R}^N$ es la imagen que se desea reconstruir.

2.7.3. Tipos de eventos

Ahora, dado que la imagen está construida a partir del recuento de fotones, es necesario clasificar adecuadamente los procesos de aniquilación positrón-electrón que, a partir de este punto, vamos a denominar eventos. Para fines prácticos, los equipos de imagen disponen de un concepto denominado *resolución temporal* esto describe a la capacidad que tiene el equipo de imagen para realizar mediciones de tiempo y, por lo tanto, de detectar cualquier señal. A partir de esto se establece una ventana de tiem-

lación el momento no es nulo. Esto obliga a considerar eventos con un margen de $0,5^\circ$ de desviación e imponiendo una restricción a la resolución de la imagen.

²²Equivalente en tres dimensiones de un pixel.

po dentro de la cual se considera que los fotones detectados provienen de un mismo evento. En la Figura 2.17 se muestra un ejemplo de la ventana de coincidencia para dos eventos.

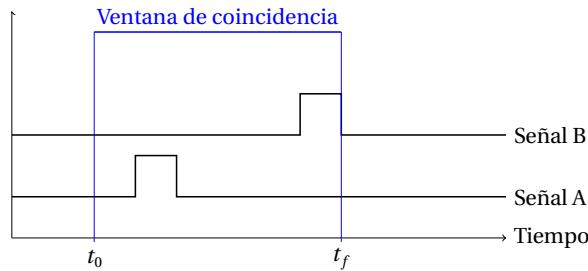


Figura 2.17: Ventana de coincidencia para dos eventos.

En consecuencia, se pueden clasificar a los eventos en las siguientes categorías:

1. **Eventos verdaderos o reales:** son aquellos eventos que se generan por la aniquilación de un positrón y un electrón en un voxel y los fotones resultantes son detectados en la ventana de coincidencia.
2. **Eventos aleatorios:** son aquellos eventos que se generan por la detección de dos fotones que no provienen del mismo evento pero son detectados en la ventana de coincidencia.
3. **Eventos aislados:** son aquellos eventos que se generan por la detección de un solo fotón en la ventana de coincidencia.
4. **Eventos dispersos:** son aquellos eventos que se generan por la detección de uno o ambos fotones del mismo evento los cuales han estado sujetos al efecto Compton²³.
5. **Eventos múltiples:** son aquellos en los que se detectan 3 fotones resultado de 2 eventos de aniquilación.

En la Figura 2.18 se muestra un ejemplo de la clasificación de eventos en una imagen PET.

²³El efecto Compton se refiere a la interacción entre un fotón y, de manera general, un electrón libre dando como resultado una modificación en la dirección y velocidad del fotón inicial. Este efecto es común en el tejido humano y es el responsable de la dispersión de los fotones. Ver [1]

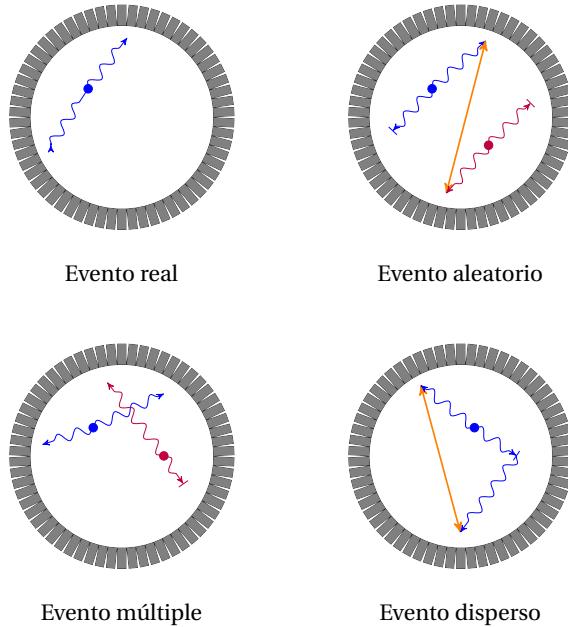


Figura 2.18: Clasificación de eventos en una imagen PET.

2.7.4. Reconstrucción de imágenes PET

En esta sección discutiremos los métodos empleados para la reconstrucción de imágenes PET. En general, podemos dividir los métodos en dos categorías: métodos analíticos y métodos iterativos. Adicionalmente, se pueden clasificar según el tipo de reconstrucción a considerar, es decir si es 2D o 3D. Más información sobre la reconstrucción de imágenes PET se puede encontrar en [1], para métodos iterativos se puede consultar [35].

Organización de los datos en 2D

Líneas de respuesta De manera general, un escáner PET realiza un conteo de los eventos detectados en cada par de detectores; de esta manera, podemos trazar una línea que conecte los detectores y que pase por el voxel en el que se generó el evento. A esta línea la denominamos *línea de respuesta* a partir de la cual se traza un pequeño volumen dado por el tamaño del detector y al cual denominaremos *tubo de respuesta*. Así, a cada par de detectores, digamos, d_a, d_b le corresponde una línea de respuesta \mathcal{L}_{d_a, d_b} y una función de sensibilidad ψ_{d_a, d_b} a partir de la cual el número de eventos coincidentes detectados en el par de detectores se puede modelar como una variable aleatoria con distribución

de Poisson con media

$$\langle p_{d_a, d_b} \rangle = \tau \int_{\text{FOV}} f(\mathbf{r}) \psi_{d_a, d_b}(\mathbf{r}) d\mathbf{r}, \quad (2.34)$$

donde $\mathbf{r} = (x, y, z)$ es la posición del voxel en el espacio, τ es el tiempo de adquisición, $f(\mathbf{r})$ es la concentración de trazador y FOV^{24} es el campo de visión del escáner. Para este caso vamos a considerar que la concentración del trazador es estacionaria y

$$f(\mathbf{r}) = 0 \text{ para } \sqrt{x^2 + y^2} > R_F,$$

donde R_F es el radio del campo de visión. Con esto, el problema se reduce a la obtención de $f(\mathbf{r})$ a partir de los datos $\langle p_{d_a, d_b} \rangle$ para cada par de detectores.²⁵

El modelo de la ecuación (2.34) asume que los efectos no lineales debido a coincidencias aleatorias y tiempo muerto²⁶ ya han sido corregidos y por lo tanto el modelo es lineal. En este sentido, si consideramos un método analítico en el que, además de las condiciones anteriores, se asume la corrección de la atenuación, entonces cada tubo de respuesta se puede modelar como una única línea en el centro del detector. De esta manera, la ecuación (2.34) se puede reescribir como

$$\langle p_{d_a, d_b} \rangle = \int_{\mathcal{L}_{d_a, d_b}} f(\mathbf{r}) d\mathbf{r}, \quad (2.35)$$

pues la función de sensibilidad está dada por

$$\psi_{d_a, d_b}(\mathbf{r}) = \begin{cases} 1 & \text{si } \mathbf{r} \in \mathcal{L}_{d_a, d_b}, \\ 0 & \text{caso contrario.} \end{cases} \quad (2.36)$$

Sinograma La información generada por el escáner PET se puede organizar de múltiples maneras de acuerdo a las necesidades de estudio y ajustándose a las restricciones de almacenamiento y capacidad de procesamiento. Una forma consiste en guardar los registros individuales de cada evento junto con la información de los detectores que lo registraron de forma secuencial, a esta forma de almacenamiento se la denomina *list-mode* y es la forma más sencilla de almacenar la información generada. No obstante,

²⁴En inglés, Field-of-View.

²⁵En [1] se menciona que para escáneres modernos, el número de detectores puede superar los 10^9 .

²⁶Se refiere al tiempo en el que el detector no puede detectar eventos debido a varios factores como limitaciones en la electrónica, procesamiento, etc.

esta forma no es eficiente en términos de procesamiento por lo que definiremos una parametrización que optimiza la forma en la que se guarda la información. Un ejemplo se puede ver en la figura 2.7.

En este sentido, se define un *sinograma* como una parametrización, alternativa a las coordenadas cartesianas, en la cual utilizaremos las variables s y θ para representar la distancia del al centro del detector y el ángulo de rotación respecto al eje de coordenadas. En la Figura 2.19 se muestra un ejemplo de la parametrización de un sinograma considerando un corte axial con $z = z_0$.

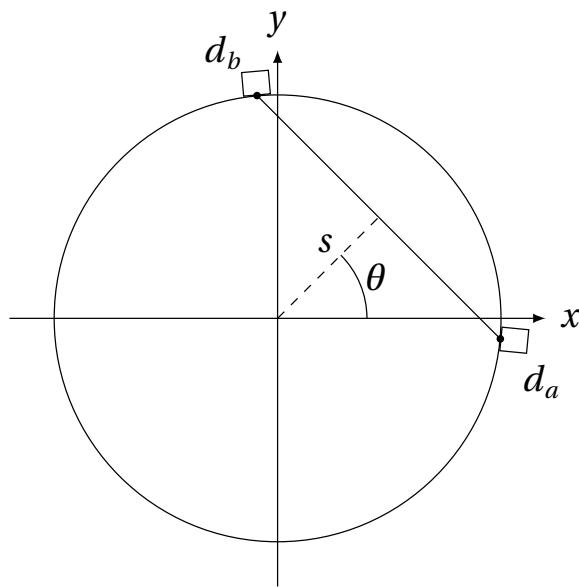


Figura 2.19: Parametrización de un sinograma.

Así, a partir de la parametrización anterior, se pueden definir las integrales de línea de la distribución del trazador de la siguiente manera

$$p(s, \theta, z_0) = \int_{-\infty}^{\infty} f(s \cos(\theta) - t \cos(\theta), s \sin(\theta) + t \sin(\theta), z_0) dt, \quad (2.37)$$

donde t representa la coordenada a lo largo de la línea de respuesta.

Ahora, definamos el operador T el cual relaciona la imagen f con el sinograma p de la siguiente manera

$$p(s, \theta) = T[f](s, \theta),$$

a este operador se lo denomina *transformada de rayos X*²⁷ y a la función p se la denomina *sinograma* a partir de la cual, se sigue que

$$p_{d_a, d_b} \approx p(s, \theta).$$

Ejemplos Para ilustrar el concepto de sinograma, que será fundamental para el proceso de reconstrucción, consideremos los siguientes ejemplos que muestran las figuras originales junto con su sinograma. En todos los casos, las imágenes originales son en escala de grises y el sinograma se generó utilizando el código A.9 junto con la librería scikit-image de Python

En la figura 2.20 se muestra un ejemplo de la parametrización de un sinograma para un círculo. Note que, en este caso, como el círculo no se encuentra en el centro del campo de visión, el sinograma no es simétrico y presenta un desplazamiento respecto al eje s conforme el ángulo θ varía.

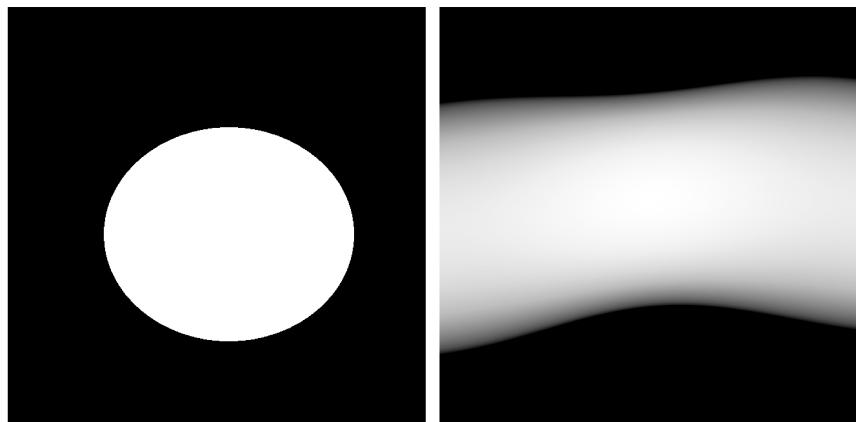


Figura 2.20: Ejemplo de un sinograma para un círculo.

De la misma manera, en la figura 2.21 se muestra un ejemplo de la parametrización de un sinograma para una figura mixta, compuesta por tres figuras geométricas. En este caso, el sinograma presenta una mayor variabilidad en la amplitud de las líneas debido a la presencia de diferentes figuras geométricas.

Finalmente, una imagen estándar utilizada para el estudio de la reconstrucción de imágenes médicas es la denominada como forma de Shepp-Logan. En la figura 2.22 se

²⁷En dos dimensiones, la transformada de rayos X es equivalente a la transformada de Radón.

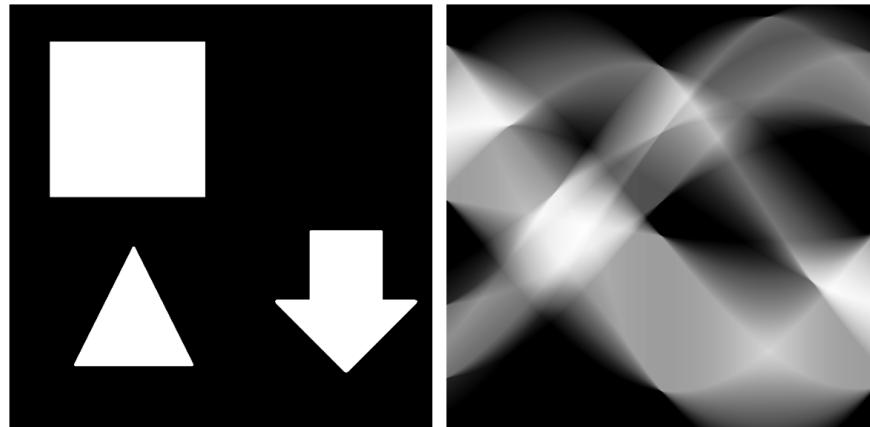


Figura 2.21: Ejemplo de un sinograma para una figura mixta.

muestra un ejemplo de la parametrización de un sinograma para la forma de Shepp-Logan.

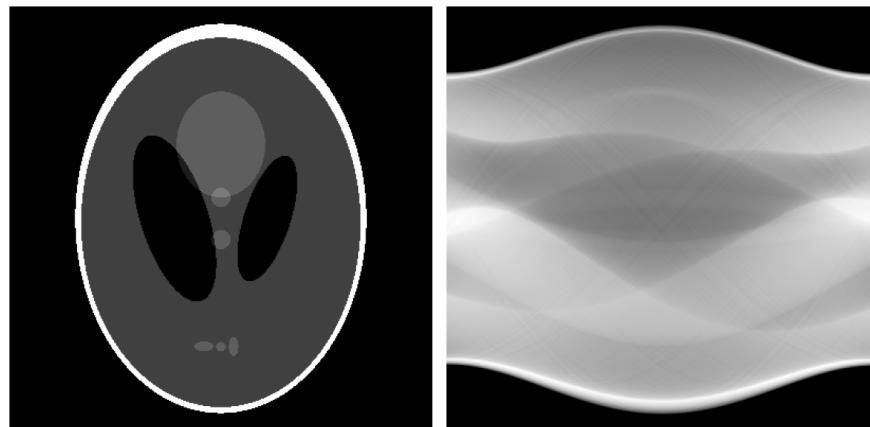


Figura 2.22: Ejemplo de un sinograma para la forma de Shepp-Logan.

Datos 2D multi-corte

En la Sección 2.7.4 se discutió sobre la organización de los datos para el caso de un único corte axial z . En esta sección vamos a considerar el caso en el que se admiten múltiples cortes. En general, los escáneres están diseñados con múltiples anillos de detectores que permiten la adquisición de múltiples cortes en un solo barrido. La figura 2.23 ilustra un esquema de un escáner PET con 4 anillos de detectores, para una mejor apreciación, uno de ellos se encuentra en color naranja.

En esta sección, también vamos a considerar los eventos detectados en los diferentes

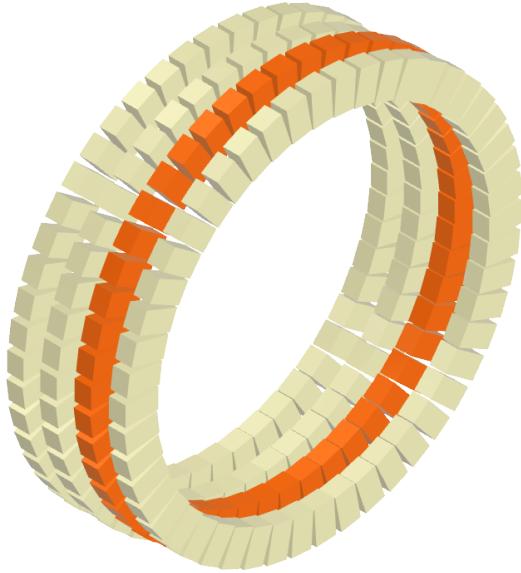


Figura 2.23: Esquema de un escáner PET con múltiples anillos de detectores.

anillos de detectores. Si denotamos como N_R a la cantidad de anillos detectores separados axialmente por una distancia Δz , entonces la cantidad de anillos está indexada de la siguiente forma $r = 0, 1, \dots, N_R - 1$. Con lo cual, los eventos detectados en el mismo anillo se denominan *coincidencias directas* y en consecuencia son organizadas en un sinograma directo notado por $p(s, \theta, z = r\Delta z)$ como se describió en las secciones previas. Por otro lado, para los eventos detectados en anillos distintos se introduce el concepto de *diferencia de anillo máxima* $d_{2D_máx}$,²⁸ esta hace referencia al número máximo de anillos a partir del anillo base empleados para la detección de eventos. De esta manera, si consideramos el anillo r_0 como base, para la detección de los eventos se admitirá los detectados en los anillos

$$r + d \quad \text{con} \quad d = -d_{2D_máx}, \dots, d_{2D_máx}. \quad (2.38)$$

Así, de manera general las LOR (*Line of reference*) entre los anillos $r - j$ y $r + j$ con $j = 0, 1, \dots, d_{2D_máx}/2$ son sumadas para formar el sinograma directo del corte $z = r\Delta z$ y las LOR entre los anillos $r - j + 1$ y $r + j$ con $j = 0, 1, \dots, (d_{2D_máx} - 1)/2$ son sumadas para formar el sinograma del corte cruzado con $z = (r + 1/2)\Delta z$. En la Figura 2.24 se muestra un ejemplo de las diferentes contribuciones de los anillos de detectores para la formación de un sinograma. Además, para ilustrar de una mejor manera esta situación, en la

²⁸En general se utiliza un valor $d_{2D_máx} = 5$.

Figura 2.25 se muestra una versión en 3D de la figura 2.24 para los eventos marcados en color negro.

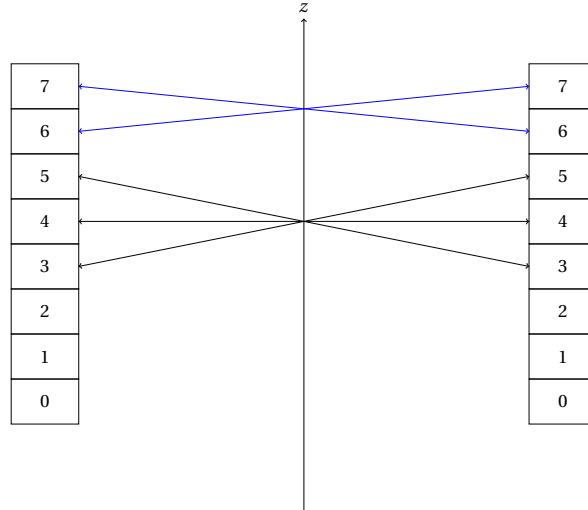


Figura 2.24: Ejemplo de las diferentes contribuciones de los anillos de detectores para la formación de un sinograma, considerando $d_{2D_máx} = 2$. En azul se muestran las LOR para un corte con $z = 13\Delta/2$ en las cuales se encuentran las contribuciones de los anillos 6 y 7. En negro se muestran las LOR para un corte con $z = 8\Delta/2$ obtenido al considerar las coincidencias entre los pares de anillos $(r_a, r_b) = (3, 5), (5, 3)$ y $(4, 4)$.

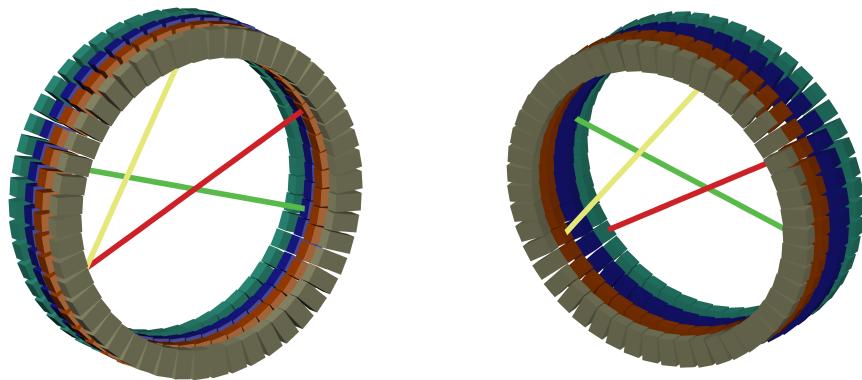


Figura 2.25: Ejemplo de un escáner con 4 anillos y 3 eventos que son detectados siguiendo el esquema presentado en la Figura 2.24. Note que el evento de color verde se detecta únicamente en el anillo de color azul. Mientras que los eventos rojo y amarillo son detectados tanto por el anillo verde como naranja.

Reconstrucción 2D

Reconstrucción iterativa En esta sección realizaremos una breve introducción a los métodos iterativos más populares son ML-EM²⁹ y OSEM³⁰ para la reconstrucción de imágenes en PET, para una revisión más detallada se puede consultar [35]. En contraste a los métodos analíticos, en los cuales se asume una continuidad en los datos; los métodos iterativos utilizan los datos de manera discreta. Además, los últimos son independientes de la geometría utilizada en la adquisición. Por lo tanto, los conceptos presentados son aplicables tanto a datos 2D y 3D.

Observación 2.14. *De aquí en adelante, denotaremos los vectores por negritas, por ejemplo \mathbf{p} .*

Modelo de datos

Para este caso utilizaremos una versión simplificada de la ecuación (2.34) en la que indexamos por j a cada par de detectores (d_a, d_b), de este modo

$$\langle \mathbf{p}_j \rangle = \tau \int_{\text{FOV}} f(\mathbf{r}) \Psi_j(\mathbf{r}) d\mathbf{r} \quad \forall j = 1, \dots, N_{\text{LOR}}. \quad (2.39)$$

Además, los datos en bruto p_j corresponden a el recuento de eventos detectados en el par de detectores j -ésimo siguiendo una distribución de Poisson. Cualquier efecto físico lineal puede ser modelado en la función de sensibilidad Ψ_j por ejemplo: atenuación, dispersión (asumiendo que se conoce el modelo de densidad), aberturas entre los detectores, etc. De esta manera, la precisión del modelo físico determinará la precisión de la reconstrucción. De esta manera, la función de *verosimilitud* está dada por

$$\Pr[\mathbf{p} | f] = \prod_{j=1}^{N_{\text{LOR}}} \frac{\exp(-\langle p_j \rangle) \langle p_j \rangle^{p_j}}{p_j!}, \quad (2.40)$$

²⁹En inglés *Maximum Likelihood Expectation Maximization*.

³⁰En inglés *Ordered Subset Expectation Maximization*.

En particular, cuando el número de eventos detectados es suficiente la función de verosimilitud se puede aproximar por una distribución normal de la siguiente manera

$$\Pr[\mathbf{p} | f] = \prod_{j=1}^{N_{\text{LOR}}} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(p_j - \langle p_j \rangle)^2}{2\sigma_j^2}\right), \quad (2.41)$$

donde la varianza σ_j de cada LOR se puede estimar a partir de los datos obtenidos. Variaciones sobre el modelo presentado se pueden encontrar en [1].

Modelo de la imagen Los métodos iterativos asumen que la imagen es una combinación lineal de funciones de base, es decir,

$$f(x, y) = \sum_{i=1}^P b_i(x, y) f_i, \quad (2.42)$$

donde las funciones de base, en la mayoría de los algoritmos son contiguas y no se superponen creando una partición del espacio. Por ejemplo:

$$b_i(x, y) = \begin{cases} 1 & \text{si } |x - x_i| < \frac{\Delta x}{2} \text{ y } |y - y_i| < \frac{\Delta x}{2}; \\ 0 & \text{si } |x - x_i| \geq \frac{\Delta x}{2} \text{ y } |y - y_i| \geq \frac{\Delta x}{2} \end{cases}, \quad (2.43)$$

donde $i = (i_x, i_y)$ y el centro del pixel i -ésimo es $(i_x \Delta x, i_y \Delta y)$ y el tamaño del pixel es $\Delta x = \frac{\Delta s}{Z}$ con Z el *factor de aumento*. Notemos que la función de base b_i no es continua, variaciones sobre la elección de esta función se pueden realizar debido a consideraciones al realizar la transformada de Fourier. En la Figura 2.26 se muestra un ejemplo simplificado del modelo de la imagen, en el mismo se muestra una imagen discreta con 9 píxeles de dimensiones 3×3 . Además, se incluyen las líneas de respuesta (LOR) que cruzan por el centro de la imagen y que se encuentran asociados a cada par de detectores que se encuentran diametralmente opuestos. No obstante, en la práctica, por un mismo pixel pueden pasar múltiples LOR que pueden proceder de distintos anillos de detectores. Ejemplos de esto se pueden ver en las Figuras 2.24 y 2.25.

Ahora, dado que la elección de la función de base determina el modelo de la imagen, el problema se reduce a estimar el vector de parámetros $\{f_i : i = 1, \dots, P\}$, usualmente con la condición de que $f_i \geq 0$ para todo i . En nuestro caso, nos concentraremos en

el *modelo Bayesiano*³¹ en el cual se asume que la imagen es una variable aleatoria y se define una distribución de probabilidad a priori $\text{Pr}[\mathbf{f}]$. En la práctica, la distribución a priori se elige empíricamente debido a la complejidad de la información que se desearía expresar de manera matemática.

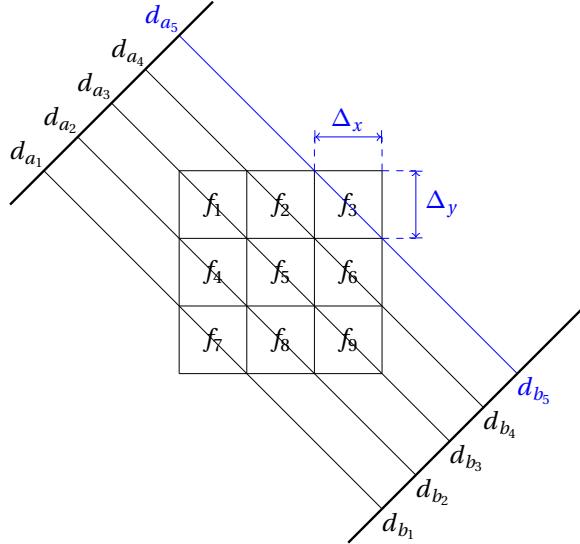


Figura 2.26: Ejemplo de una imagen discreta.

Matriz del sistema Para resumir el modelo de datos y el modelo de la imagen, consideremos las ecuaciones (2.39) y (2.42). El problema se reduce al siguiente sistema de ecuaciones lineales

$$\langle \mathbf{p}_j \rangle = \sum_{i=1}^P a_{j,i} f_i \quad \forall j = 1, \dots, N_{\text{LOR}}, \quad (2.44)$$

donde los elementos de la matriz del sistema $a_{j,i}$ están dados por

$$a_{j,i} = \tau \int_{\text{FOV}} b_i(\mathbf{r}) \Psi_j(\mathbf{r}) d\mathbf{r} \quad \forall j = 1, \dots, N_{\text{LOR}} \quad \text{y} \quad \forall i = 1, \dots, P. \quad (2.45)$$

De esta manera, si la integral del modelo anterior solo incluye corrección de atenuación y normalización entonces la matriz del sistema es de tipo *sparse* con elementos que pueden ser calculados de manera eficiente durante el proceso. No obstante, en general, una inversión del sistema de la ecuación (2.44) no es posible debido a las siguientes razones:

³¹Para más información sobre el modelo Bayesiano ver [32]

1. El sistema es *mal condicionado* debido a que el número de condición de A es demasiado grande. En consecuencia, pequeñas variaciones en los datos pueden llevar a grandes variaciones en la solución.³²
2. Dado que para un equipo moderno $P \approx 10^6$ y $N_{\text{LOR}} \approx 10^9$ la inversión de la matriz es computacionalmente costosa.

Para resolver el primer inconveniente se puede utilizar incorporando información a priori en la función de costo. El segundo problema puede ser resuelto optimizando la función de costo por aproximaciones sucesivas.

Función de costo Una parte fundamental para los algoritmos iterativos es la función de costo que denotaremos $Q(\mathbf{f}, \mathbf{p})$ y que depende de los coeficientes de la imagen y de los datos medidos, a esta función también se la denomina función objetivo. En general, la imagen reconstruida estima f^* y se define como

$$\mathbf{f}^* = \arg \min_{\mathbf{f}} -Q(\mathbf{f}, \mathbf{p}). \quad (2.46)$$

con la condición usual de que $f_i \geq 0$ para todo i . Desde un punto de vista Bayesiano, la función de costo es la *distribución de probabilidad a posteriori* y se define como

$$\Pr[\mathbf{f} | \mathbf{p}] = \frac{\Pr[\mathbf{p} | \mathbf{f}] \Pr[\mathbf{f}]}{\Pr[\mathbf{p}]} \quad (2.47)$$

A partir de la ecuación anterior se puede definir la función de costo como el logaritmo de la distribución de probabilidad a posteriori, es decir

$$Q(\mathbf{f}, \mathbf{p}) = \log \Pr[\mathbf{p} | \mathbf{f}] + \log \Pr[\mathbf{f}] \quad (2.48)$$

De esta manera, el primer término penaliza el hecho de que la imagen que se está obteniendo no se ajuste adecuadamente a los datos mientras que el segundo término *estabiliza* la inversión penalizando las imágenes que son a priori improbables. Dado que buscamos que la solución encontrada sea única, la función de costo debe ser convexa y diferenciable, para esto basta con verificar que la matriz Hessiana de la función de costo sea definida negativa para todos los vectores \mathbf{f} factibles.

³²Para más información sobre el número de condición de una matriz ver [34] Capítulo 2.

Algoritmos de optimización Gracias a la definición de la función de costo en la sección anterior, basta determinar un algoritmo de optimización que nos permita encontrar una sucesión de imágenes \mathbf{f}_n con $n = 1, 2, \dots$ tal que

$$\lim_{n \rightarrow +\infty} \mathbf{f}_n = \mathbf{f}^*. \quad (2.49)$$

Además de la convergencia asintótica el algoritmo seleccionado debe ser estable, eficiente y rápido e independiente de la elección inicial de la imagen \mathbf{f}_0 . Más aún, la convergencia monótona es deseable pues gracias a ésta, en algunos casos, se puede demostrar la convergencia asintótica.

En general, cuando la función de costo es diferenciable y la solución buscada debe ser no negativa entonces la solución \mathbf{f}^* satisface las condiciones de *Karush-Kuhn-Tucker* (KKT):

$$(\nabla Q(\mathbf{f}^*, \mathbf{p}))_j = 0, \quad \text{si } f_j^* > 0, \quad \forall j = 1, \dots, P, \quad (2.50)$$

$$(\nabla Q(\mathbf{f}^*, \mathbf{p}))_j \leq 0, \quad \text{si } f_j^* = 0, \quad \forall j = 1, \dots, P, \quad (2.51)$$

donde el gradiente de la función de costo se define como

$$(\nabla Q(\mathbf{f}^*, \mathbf{p}))_j = \left. \frac{\partial Q(\mathbf{f}, \mathbf{p})}{\partial f_j} \right|_{\mathbf{f}=\mathbf{f}^*}. \quad (2.52)$$

Cuando la condición de positividad no es necesaria, las condiciones KKT se reducen a la ecuación (2.50). En la siguiente lista se describen algunos los algoritmos más populares:

1. Métodos basados en el gradiente.
 - a) Descenso del gradiente.
 - b) Descenso del gradiente estocástico.
2. Métodos basados en subconjuntos del vector imagen.
 - a) Descenso coordinado
3. Métodos basados en substituir la función de costo.
 - a) Maximización de la verosimilitud (ML-EM).

- b) Mínimos cuadrados ISRA.
4. Métodos basados en bloques iterativos.
- a) Maximización de la verosimilitud en subconjuntos ordenados (OSEM).

Descripciones más detalladas de los algoritmos se pueden encontrar en [1, 4, 34].

ML-EM y OSEM El método ML-EM fue introducido por Dempster en 1977 [12] y aplicado a PET por Shepp y Vardi en 1982 [37] y Lange y Carson en 1984 [25]. No obstante, el método OSEM introducido por Hudson y Larkin en 1994 fue el primer método iterativo en ser ampliamente utilizado en la práctica clínica.

La función de costo para los métodos ML-EM y OSEM está basada en la ecuación (2.39). Reemplazando la ecuación (2.44) en la ecuación (2.39) y manipulando convenientemente se obtiene la siguiente función de costo

$$Q(\mathbf{f}, \mathbf{p}) = \sum_{j=1}^{N_{\text{LOR}}} \left(- \sum_{i=1}^P a_{j,i} f_i + p_j \log \left(\sum_{i=1}^P a_{j,i} f_i \right) \right). \quad (2.53)$$

Y la actualización de la imagen reconstruida se realiza de la siguiente manera

$$\mathbf{f}_i^{(n+1)} = f_i^{(n)} \frac{1}{\sum_{k=1}^{N_{\text{LOR}}} a_{k,i}} \frac{\sum_{j=1}^{N_{\text{LOR}}} a_{j,i} p_j}{\sum_{l=1}^P a_{j,l} f_l^{(n)}} \quad \forall i = 1, \dots, P. \quad (2.54)$$

De esta manera, el algoritmo ML-EM tiene las siguientes propiedades relevantes:

1. La función de costo se incrementa de manera monótona en cada iteración, es decir $Q(\mathbf{f}^{(n+1)}, \mathbf{p}) \geq Q(\mathbf{f}^{(n)}, \mathbf{p})$.
2. Las iteraciones $\mathbf{f}^{(n)}$ convergen a la solución óptima \mathbf{f}^* la cual maximiza la función de costo.
3. El algoritmo puede ser implementado fácilmente para datos en modo lista, ver [3].

El algoritmo OSEM [20] es una variante del método ML-EM obtenido a partir de una modificación en la ecuación (2.54) en la cual la información de las líneas de respuesta

se partitiona en S subconjuntos disjuntos $J_1, \dots, J_S \subseteq [1, \dots, N_{\text{LOR}}]$ y se actualiza la imagen reconstruida de la siguiente manera

$$f_i^{(n+1)} = f_i^{(n)} \frac{1}{\sum_{k \in J_n \text{ mód } S} a_{k,i}} \sum_{j \in J_n \text{ mód } S} \frac{a_{j,i} p_j}{\sum_{l=1}^P a_{j,l} f_l^{(n)}} \quad \forall i = 1, \dots, P. \quad (2.55)$$

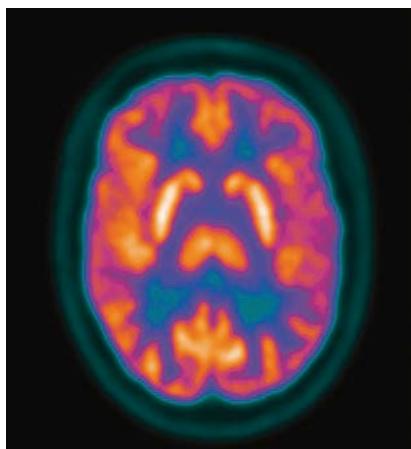
De forma empírica, se ha observado que el método OSEM converge en un factor de aproximadamente S más rápido que el método ML-EM. Además, algunos autores sugieren reducir progresivamente el número de subconjuntos durante cada iteración.

2.7.5. Ejemplos prácticos de uso médico

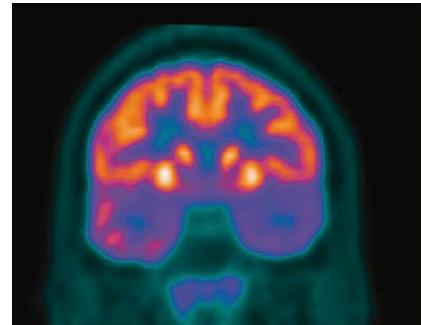
En esta sección realizaremos una breve descripción de la aplicación práctica del uso de imágenes PET/CT y PET/MR en la medicina nuclear. Note que en la práctica, los escáneres utilizados poseen una combinación de técnicas funcionales y anatómicas, debido a la necesidad de identificar la ubicación y actividad metabólica de manera simultánea. Para una descripción más amplia sobre el funcionamiento de las técnicas CT y MR, que escapan a los alcances de este trabajo, se recomienda la lectura de [26].

Ahora, dado que los datos utilizados para la experimentación son de un estudio cerebral, no enfocaremos en las aplicaciones para este órgano. La información presentada en esta sección fue obtenida de [16], por lo tanto se recomienda la lectura de este documento para una descripción más detallada.

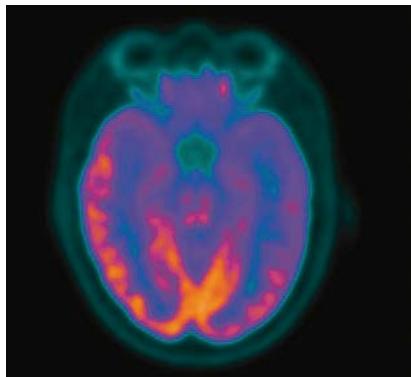
Para el primero ejemplo, consideremos las imágenes presentadas en la Figura 2.27. En este caso las imágenes PET muestran un hipometabolismo en el lóbulo temporal izquierdo, es decir una menor actividad metabólica en comparación con el lóbulo temporal derecho. Para obtener esta conclusión el estudio utilizó el compuesto ^{18}F -FDG, el cual es un análogo de la glucosa que se acumula en las células con alta actividad metabólica. De esta manera, dado que a una mayor acumulación del fármaco una mayor actividad metabólica y en consecuencia una mayor detección de los fotones emitidos, se puede concluir que la menor acumulación en el lóbulo temporal izquierdo corresponde a una menor actividad metabólica. En este primer ejemplo, se evidencia la importancia funcional de las imágenes PET en comparación con la utilidad anatómica de las imágenes CT.



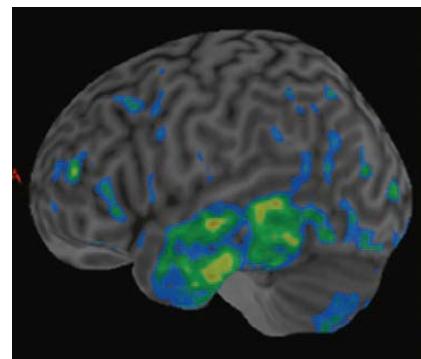
(a) Corte axial.



(b) Corte coronal.



(c) Corte temporal.

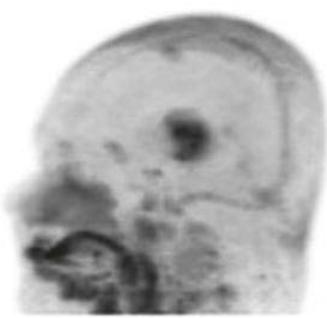


(d) Renderización 3D.

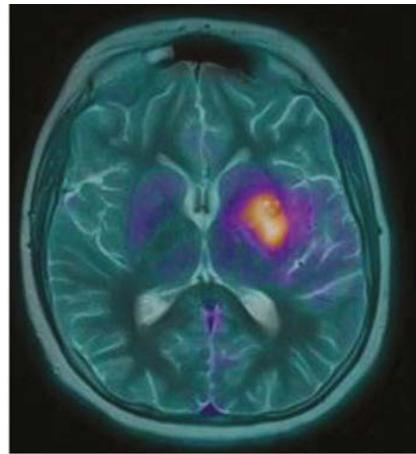
Figura 2.27: Imágenes PET/CT mostrando hipometabolismo en el lóbulo temporal izquierdo. Imágenes obtenidas de [16].

Para el segundo ejemplo consideremos la Figura 2.28, en este caso se muestra un estudio de un paciente con glioma³³ en el tálamo izquierdo. En este caso, las imágenes fueron obtenidas utilizando el fármaco ¹⁸G-DOPA y nos permiten obtener la ubicación del tumor en base a su metabolización del fármaco utilizado.

³³Tipo de tumor cerebral

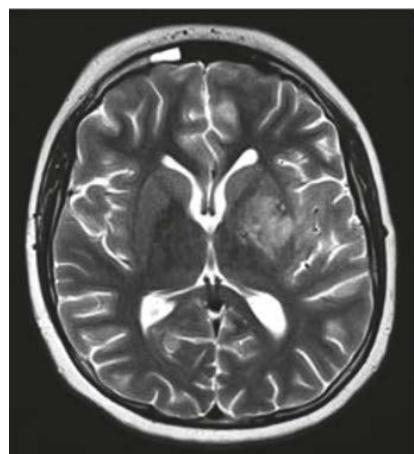


(a) Proyección de máxima intensidad.

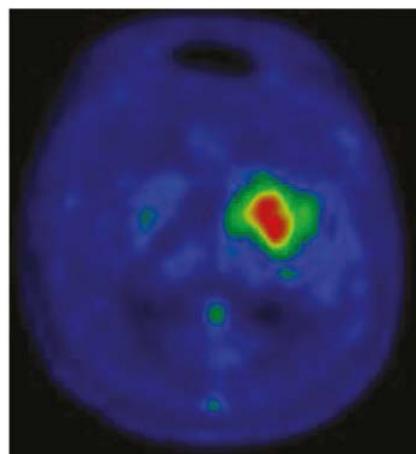


(b) Corte axial PET/MR.

Figura 2.28: Imágenes PET/MR de un paciente con glioma en el tálamo izquierdo.
Imágenes obtenidas de [16].



(c) Corte axial con MR.



(d) Corte axial con PET.

Figura 2.28: Imágenes PET/MR de un paciente con glioma en el tálamo izquierdo.
Imágenes obtenidas de [16].

Capítulo 3

Experimentación

3.1. Hardware y software

La experimentación numérica se realizó en dos computadoras con la primera una computadora portátil que denominaremos como *CP1* y la segunda una máquina virtual alojada en Azure que denominaremos como *CP2*. En la Tabla 3.1 se presentan las características de ambas computadoras. Por otra parte, en la Tabla 3.2 se presentan las librerías empleadas para la experimentación.

| Característica | CP1 | CP2 |
|------------------------|---|---|
| Modelo | Asus VivoBook 15 M513UA | Standard NC8as T4 v3 |
| Procesador | AMD Ryzen 7 5700U with Radeon Graphics 1.80 GHz | AMD EPYC 7V12 (8 núcleos) |
| GPU | AMD Radeon Graphics (Integrada) | NVIDIA Tesla T4 |
| RAM | 20 GB | 56 GB |
| Almacenamiento | 1 TB SSD (Sistema y módu- los) Azure Blob Storage (da- tos) | 128 GB SSD (Sistema y mó- dulos) / Azure Blob Storage (datos) |
| Sistema Operati- vo | Windows 11 Pro | Ubuntu 20.04 LTS |

Tabla 3.1: Características de las computadoras utilizadas en la experimentación.

| Librería | Versión |
|----------|---------|
| CIL | 23.1.0 |
| NiftyPET | 1.4.0 |

Tabla 3.2: Librerías utilizadas en la experimentación.

3.1.1. Datos

Para la experimentación se utilizaron varios conjuntos de datos, en la Tabla 3.3 se presentan los conjuntos de datos utilizados. En algunos casos, solo se emplearon algunos de los archivos disponibles en el conjunto de datos, por ejemplo, en el siguiente repositorio se encuentran datos para el uso de CIL: [HDTomo TXRM micro-CT datasets](#).

| Código | Nombre | Descripción | Fuente |
|--------|----------------------------|--|--------|
| D0 | Micro TC USB | Micro tomografías computarizadas realizadas a 6 objetos en formato TXRM/TXM. (Se seleccionó el objeto usb) | [22] |
| D1 | Egg1 | Micro tomografías computarizadas realizadas a 6 objetos en formato TXRM/TXM. (Se seleccionó el objeto egg1: Kinder Surprise chocolate egg in aluminium foil with a plastic container holding a toy and assembly instructions inside) | [22] |
| D2 | Escáner PET amiloide único | Datos PET en modo lista de 60 minutos de adquisición, utilizando un trazador de amiloide, florbetapir, proporcionado por Eli Lilly | [27] |

Tabla 3.3: Datos utilizados en la experimentación.

Finalmente, los archivos de scripts, cuadernos y consolidados de resultados se encuentran disponibles en el siguiente repositorio: [Github: Reconstrucción de imágenes PET con NiftyPET](#).

3.2. Core Image Libray (CIL)

En esta sección, realizaremos experimentaciones con CIL (Core Imaging Library), una plataforma avanzada que proporciona herramientas para la reconstrucción de imágenes en diversas modalidades de imagen médica, incluyendo la tomografía computarizada (CT). Dado que los problemas de reconstrucción de imágenes en tomografía computarizada son similares a los que se presentan en la tomografía por emisión de positrones (PET), utilizaremos ejemplos de CT para ilustrar el proceso. Esto servirá como preparación para abordar el trabajo específico con imágenes PET.

CIL es una herramienta potente que facilita la implementación y prueba de algoritmos de reconstrucción de imágenes, proporcionando una plataforma flexible y extensible para investigadores y desarrolladores. Para obtener más información sobre el uso de esta biblioteca y acceder a la documentación de la última versión estable disponible al momento de realizar este trabajo, se puede consultar el siguiente enlace: [documentación CIL](#).

3.2.1. Definición de geometrías

En esta sección vamos a revisar la definición de geometrías para su uso en CIL dado que, aunque la mayoría de datos incluyen los metadatos necesarios, en algunos casos se puede requerir una definición manual de estas.

AcquisitionGeometry

La clase `AcquisitionGeometry` nos permite definir los parámetros necesarios que especifican la geometría de los datos. Esta posee los siguientes métodos:

1. `create_Cone2D`
2. `create_Cone3D`
3. `create_Parallel2D`
4. `create_Parallel3D`

En las Figuras 3.3 y 3.4 se muestran los ejemplos de geometrías paralelas para los casos 2D y 3D respectivamente.

Geometrías muy comunes como la de abanico 2D y cono 3D también pueden ser definidas con los métodos `create_Cone2D` y `create_Cone3D` respectivamente. En la Figura 3.5 se muestra un ejemplo de geometría de abanico 2D y en la Figura 3.6 se muestra un ejemplo de geometría de cono 3D. Por otro lado, para una mejor comprensión de las geometrías, en la Figura 3.1 se muestra una representación 3D de una geometría abanico 2D. En la misma se puede notar como se modifica la radiación emitida de manera que se forma un abanico de rayos que inciden en el objeto, para posteriormente ser detectados y finalmente reconstruir el objeto a partir de esta proyección.

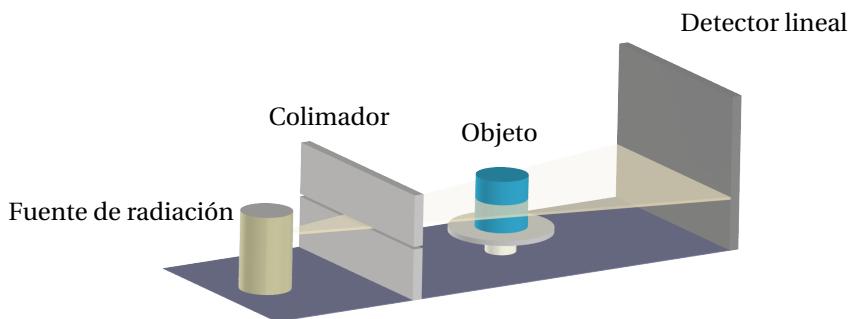


Figura 3.1: Representación 3D de una geometría abanico 2D.

Por otro lado, en la Figura 3.2 se muestra una representación 3D de una geometría cono 3D. Note que para este caso no se utiliza un colimador, sino que la radiación se emite en forma de cono y se detecta con un sensor planar. Además, de acuerdo a las dimensiones del equipo es posible que algunas partes del objeto no sean cubiertas por la radiación o en otros casos la estructura utilizada para colocar el objeto puede interferir con la radiación.

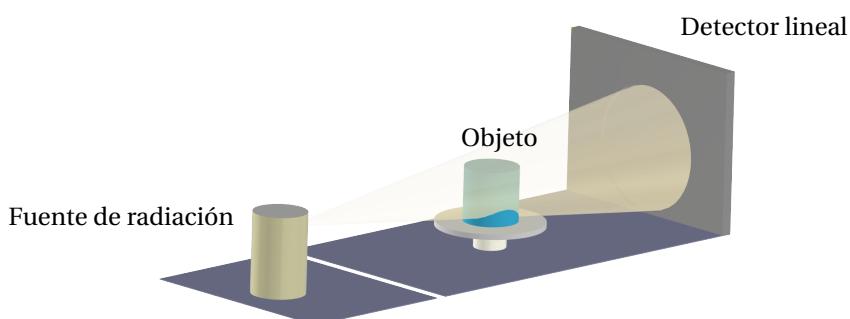


Figura 3.2: Representación 3D de una geometría cono 3D.

Dado que los entornos prácticos de funcionamiento de los equipos suelen presentar variaciones en la geometría, es posible definir geometrías que incluyan términos de desfase o rotación en la posición de la fuente, el objeto o el detector.

Finalmente, los ejemplos presentados fueron generados con el código A.10. Para más ejemplos de las geometrías disponibles en CIL se puede revisar el siguiente enlace: [Ejemplos de geometrías en CIL](#).

| | | |
|-----------------------------------|-----------------------------|-------------------------|
| — world coordinate system | ● rotation axis position | ● detector position |
| - - - ray direction | — rotation axis direction | — detector direction |
| ... image geometry | x data origin (voxel 0) | ... detector |
| - - - rotation direction θ | - - - data origin (pixel 0) | x data origin (pixel 0) |

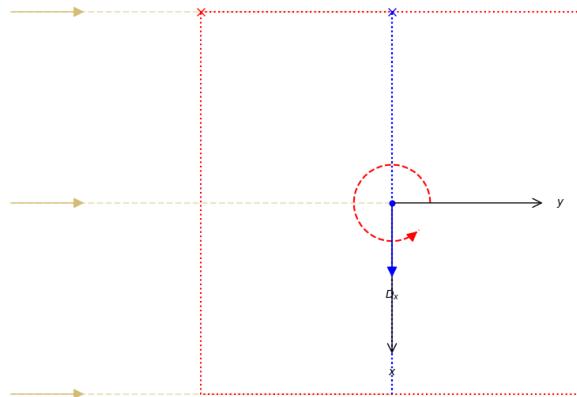


Figura 3.3: Geometría 2D.

3.2.2. Reconstrucción a partir de datos sin procesar

En la sección 3.2.1 se revisó la definición manual de geometrías en CIL. En esta sección utilizaremos el conjunto de datos D0 para realizar una reconstrucción a partir de datos sin procesar. Los siguientes resultados se obtuvieron a partir del código A.11. Primero, la geometría obtenida de los datos es la siguiente:

```
3D Cone-beam tomography
System configuration:
  Source position: [ 0.          , -95.04837036,   0.          ]
  Rotation axis position: [0., 0., 0.]
  Rotation axis direction: [0., 0., 1.]
  Detector position: [ 0.          , 55.08220291,   0.          ]
  Detector direction x: [1., 0., 0.]
  Detector direction y: [0., 0., 1.]
```

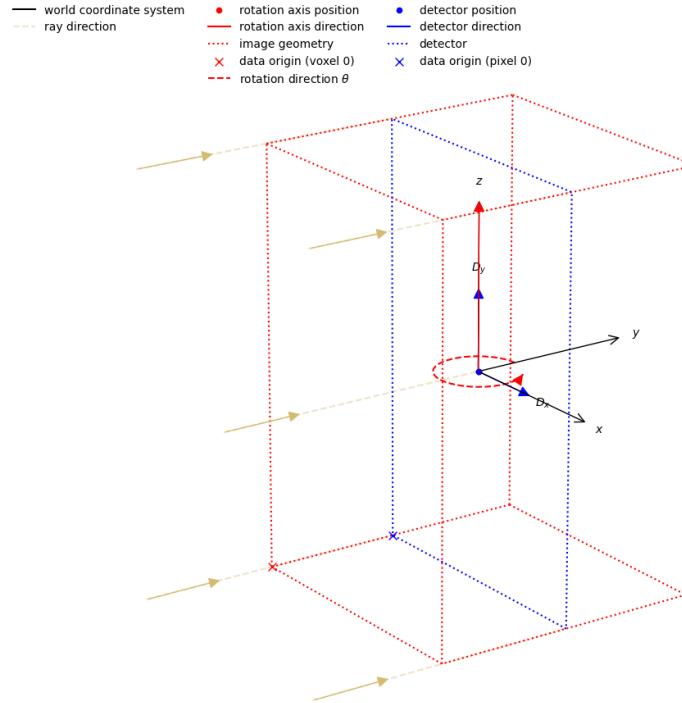


Figura 3.4: Geometría 3D.

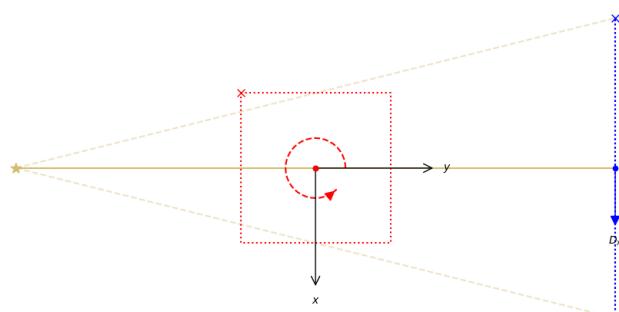
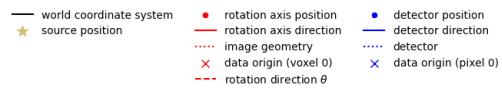


Figura 3.5: Geometría de abanico 2D.

Panel configuration:

Number of pixels: [1024 1024]

Pixel size: [0.06585435 0.06585435]

Pixel origin: bottom-left

Channel configuration:

Number of channels: 1

Acquisition description:

| | | |
|---------------------------|-----------------------------------|-------------------------|
| — world coordinate system | ● rotation axis position | ● detector position |
| ★ source position | — rotation axis direction | — detector direction |
| | ··· image geometry | ··· detector |
| | ✗ data origin (voxel 0) | ✗ data origin (pixel 0) |
| | - - - rotation direction θ | |

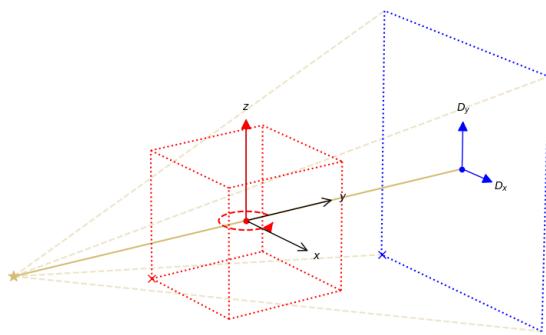


Figura 3.6: Geometría de cono 3D.

```

Number of positions: 801
Angles 0-20 in radians:
[-3.1415493, -3.1337836, -3.125914, -3.1180005, -3.1102147,
-3.1023476, -3.0945017, -3.0866468, -3.0787883, -3.0709336,
-3.0630753, -3.0552294, -3.0473533, -3.0395627, -3.0316472,
-3.0238035, -3.0159593, -3.0080967, -3.0002263, -2.9924133]
Distances in units: units distance

```

La gráfica de la geometría de los datos de este ejemplo se puede ver en la Figura 3.7.

| | | |
|---------------------------|-----------------------------------|-------------------------|
| — world coordinate system | ● rotation axis position | ● detector position |
| ★ source position | — rotation axis direction | — detector direction |
| | ··· image geometry | ··· detector |
| | ✗ data origin (voxel 0) | ✗ data origin (pixel 0) |
| | - - - rotation direction θ | |

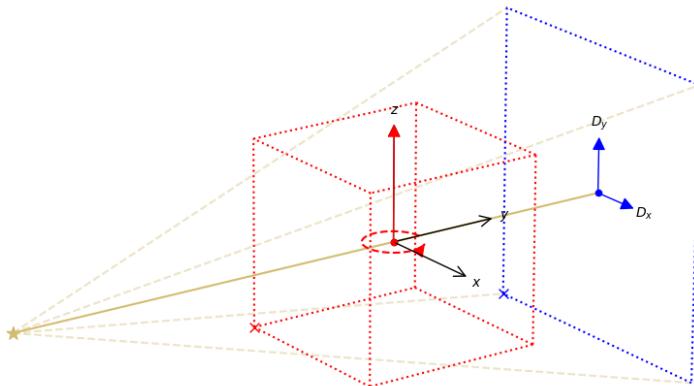


Figura 3.7: Geometría de los datos sin procesar.

Una vez cargados los datos, obtengamos el sinograma de estos sin procesar, en la

Figura 3.8 se muestra uno de los sinogramas obtenido de dimensión 801×1024 , es decir, de 801 proyecciones (ángulos) con 1024 píxeles cada una a partir de las proyecciones de una memoria USB.

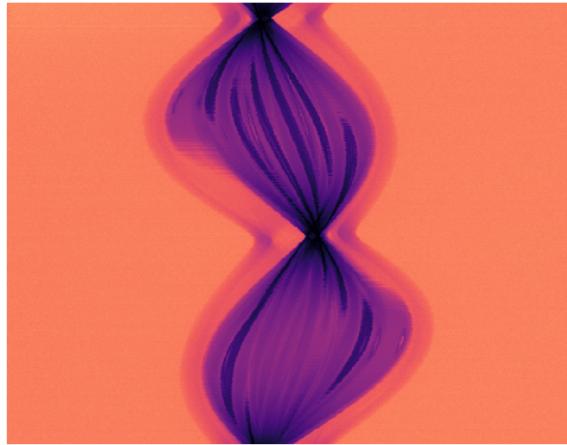


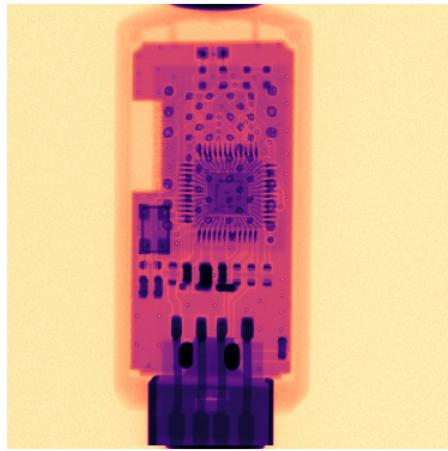
Figura 3.8: Sinograma de los datos sin procesar.

Más aún, es posible visualizar los datos sin procesar como una proyección, en la Figura 3.9a se muestra la proyección en modo *transmisión* de los datos sin procesar correspondiente al sinograma mostrado en la Figura 3.8. Por otro lado, una vez convertidos los datos de transmisión a absorción, se obtiene la proyección mostrada en la Figura 3.9b. En ambos casos las imágenes son de 1024×1024 píxeles; de hecho, hasta este punto se pueden considerar simplemente como imágenes de rayos X.

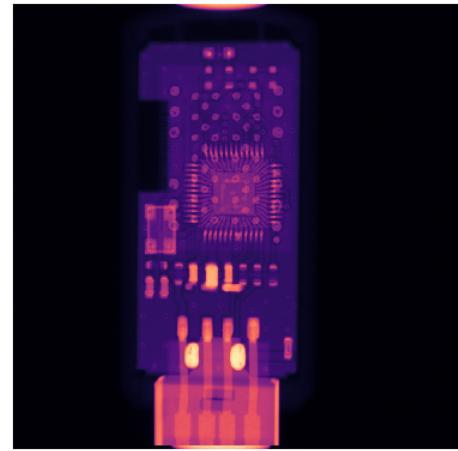
Una vez aplicado el algoritmo de reconstrucción FDK, se obtiene la reconstrucción del objeto, de modo que ahora podemos visualizar cortes transversales del objeto en varios planos y así analizar la estructura interna sin necesidad de abrirlo. En la Figura 3.10 se muestran ejemplos de los cortes obtenidos a partir de la reconstrucción del objeto. Note que en la Figura 3.10a se muestra aprecia un corte lateral mientras que en la Figura 3.10b se muestra un corte superior pudiendo observar la estructura del conector USB. Finalmente, en ambos casos las imágenes tienen una resolución de 1024×1024 píxeles y para una mejor visualización se limitaron los valores de intensidad superiores a $-0,1$.

3.2.3. Preprocesamiento de datos

Dado que en la práctica los datos no siempre se encuentran listos para realizar la reconstrucción, es necesario realizar un preprocesamiento de los datos. De esta manera,

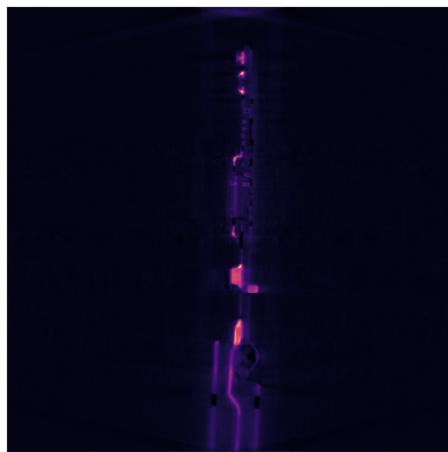


(a) Proyección de los datos en modo transmisión.

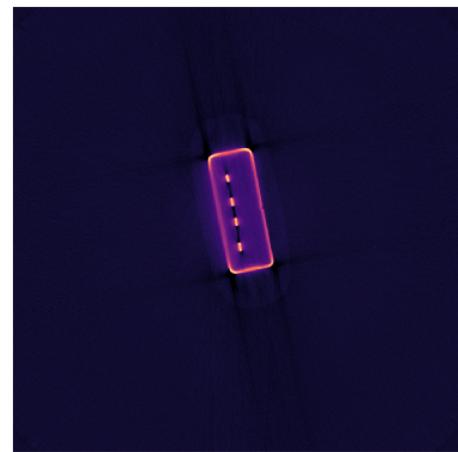


(b) Proyección de los datos convertidos a absorción.

Figura 3.9: Proyecciones de los datos sin procesar.



(a) Corte sagital.



(b) Corte transversal.

Figura 3.10: Reconstrucción FDK.

CIL nos proporciona herramientas que nos permiten entre otras cosas:

1. Compensar el desplazamiento del centro de rotación,
2. Obtener subconjuntos de datos,
3. Remover artefactos.

En el código A.12 se encuentran los pasos que se van a desarrollar en esta sección. En este caso, se utilizaron datos de ejemplo disponibles en CIL para demostrar el uso de

las herramientas de preprocessamiento, SYNCHROTRON_PARALLEL_BEAM_DATA. A continuación se muestra la información de la geometría de los datos y en la Figura 3.11 se muestra la geometría de los datos.

```
3D Parallel-beam tomography
System configuration:
Ray direction: [0., 1., 0.]
Rotation axis position: [0., 0., 0.]
Rotation axis direction: [0., 0., 1.]
Detector position: [0., 0., 0.]
Detector direction x: [1., 0., 0.]
Detector direction y: [0., 0., 1.]
Panel configuration:
Number of pixels: [160 135]
Pixel size: [1. 1.]
Pixel origin: bottom-left
Channel configuration:
Number of channels: 1
Acquisition description:
Number of positions: 91
Angles 0-20 in degrees:
[-88.2 , -86.2 , -84.2001, -82.2 , -80.2 ,
-78.2 , -76.1999, -74.2 , -72.1999, -70.2 ,
-68.2 , -66.2 , -64.1999, -62.2 , -60.2 ,
-58.2 , -56.2 , -54.2 , -52.2 , -50.2 ]
Distances in units: units distance
```

Notemos que la geometría usada corresponde a una tomografía en paralelo 3D revisada en la sección 3.2.1.

| | | |
|---------------------------|-----------------------------------|-----------------------------|
| — world coordinate system | ● rotation axis position | ● detector position |
| - - - ray direction | — rotation axis direction | — detector direction |
| ... image geometry | - - - detector | - - - data origin (voxel 0) |
| ✗ data origin (voxel 0) | - - - rotation direction θ | ✗ data origin (pixel 0) |

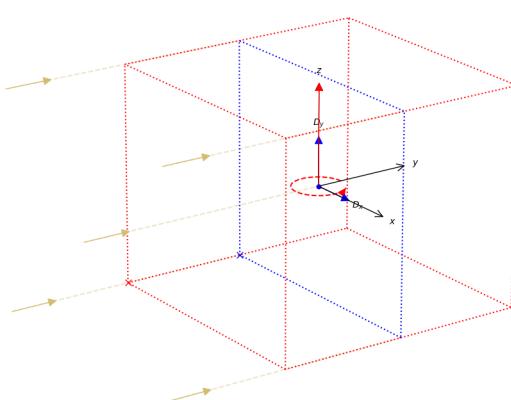


Figura 3.11: Geometría de los datos para preprocessamiento.

En la Figura 3.12 se muestra la proyección de los datos sin procesar de dimensiones 135×160 píxeles.

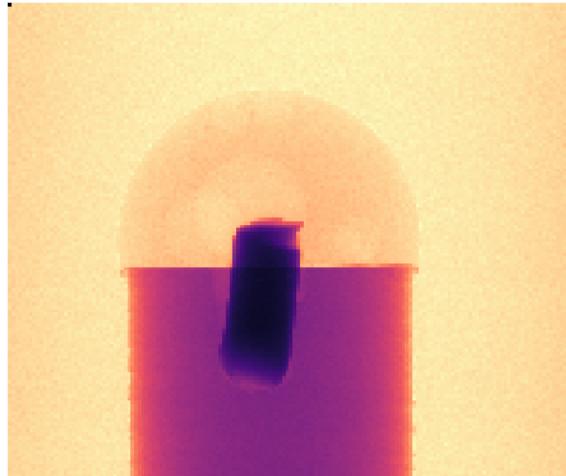


Figura 3.12: Proyección de los datos sin procesar.

Conversión de transmisión a absorción

Una vez hemos cargado los datos, dado que los datos de fondo son menores a 1, reescalamos los valores de intensidad calculando el promedio de una porción vacía y dividiendo por esta a los datos. En la Figura 3.13 se muestra la proyección de los datos convertidos a absorción, con dimensiones 135×160 píxeles en ambos casos. Una vez



(a) Proyección para el ángulo 15.

(b) Proyección para el ángulo 30.

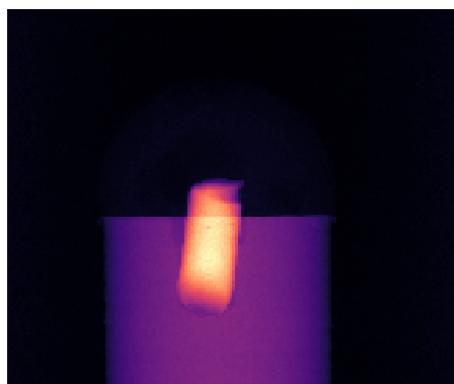
Figura 3.13: Proyección de los datos convertidos a absorción.

realizada la conversión de los datos observamos que el contraste no es el adecuado pues

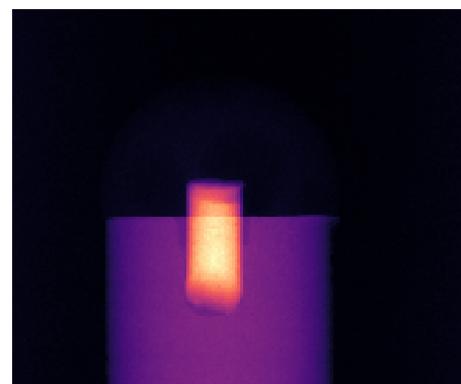
a penas se puede distinguir el objeto del medio que lo rodea. Por lo tanto, en la siguiente etapa aplicaremos un proceso simple para la corrección del contraste de la imagen.

Eliminación de pixeles brillantes

Notemos que en la Figura 3.12 la esquina superior izquierda contiene un único pixel con un valor de intensidad muy alto que, al convertir los datos a absorción, afecta el contraste de la imagen. Por lo tanto, la idea es eliminar estos pixeles brillantes para mejorar el contraste de la imagen. En la Figura 3.14 se muestra el resultado de la eliminación del pixel brillante mencionado anteriormente.



(a) Proyección corregida para el
ángulo 15.



(b) Proyección corregida para el
ángulo 30.

Figura 3.14: Proyección de los datos corregidos.

Una aproximación más elaborada pero con resultados similares al método descrito anteriormente consiste en el manejo de estas situaciones como determinación de valores atípicos. En el código A.12 se muestra cómo se puede realizar esto en la sección de eliminación de pixeles brillantes.

Corrección avanzada de pixeles defectuosos

En la sección anterior revisamos cómo se puede realizar la eliminación de pixeles brillantes. Ahora, consideremos un caso ligeramente más complejo en el cual se han producido artefactos en la imagen. La función definida en A.13 nos permite agregar artefactos a la imagen para evaluar los resultados. Así, en la Figura 3.15a se muestra un

sinograma sin artefactos mientras que en la Figura 3.15b se han agregado intencionalmente para analizar su efecto en la reconstrucción. En ambos casos los sinogramas son de 90 ángulos y 120 píxeles pues se ha tomado una porción de los datos originales que centra el objeto para una mejor visualización. Si realizamos la reconstrucción utilizando

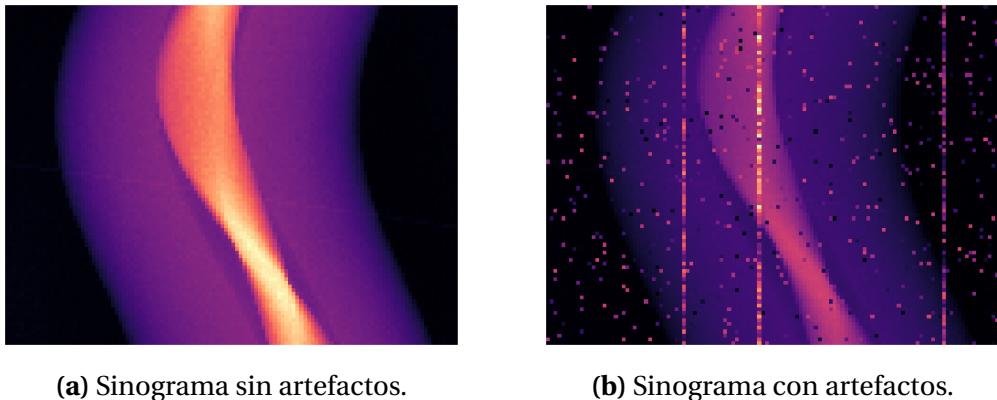


Figura 3.15: Proyección de los datos con artefactos.

estos datos junto con el algoritmo FBP, obtenemos la imagen mostrada en la Figura 3.16. Note que la imagen derecha corresponde a la reconstrucción de los datos con artefactos. En ambos casos, las imágenes obtenidas tienen una resolución de 120×120 píxeles.

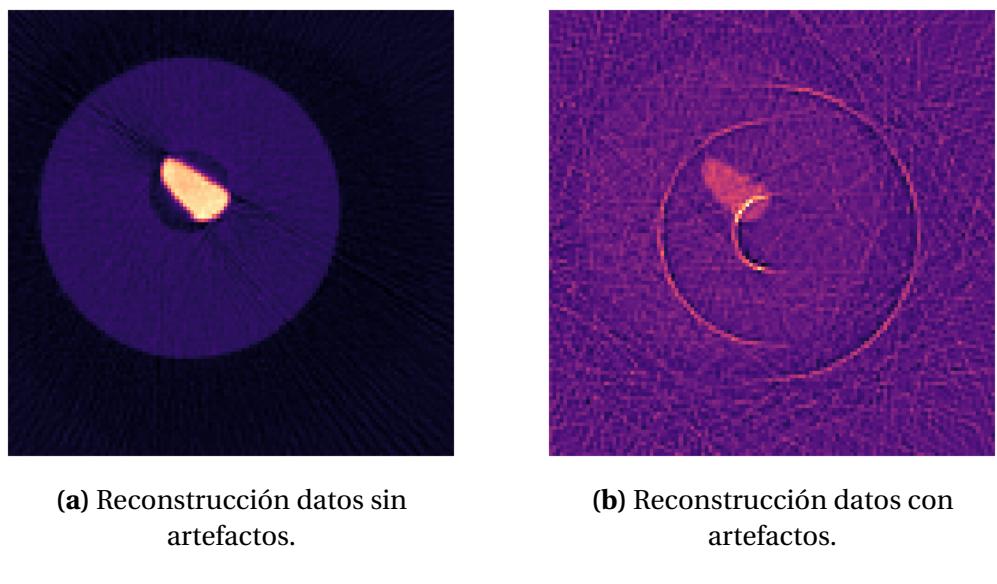


Figura 3.16: Comparativa de reconstrucción FBP con artefactos.

Para corregir esto, podemos utilizar la clase `MaskGenerator` que nos permite definir un umbral para la detección de valores atípicos así como una ventana para la interpola-

ción de los valores. En la Figura 3.17 se muestra la máscara generada para la detección de valores atípicos.

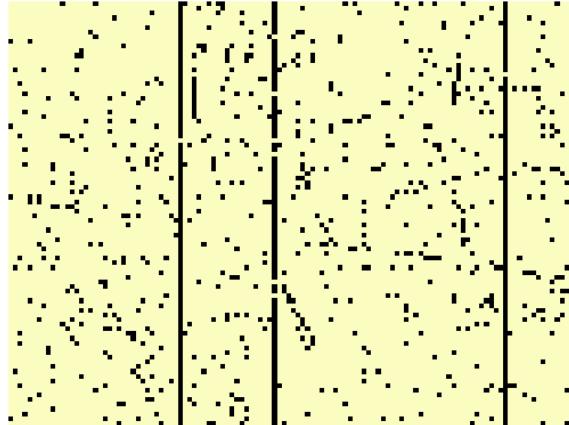
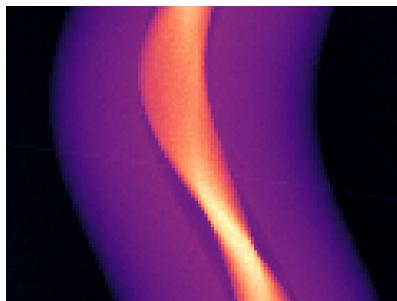
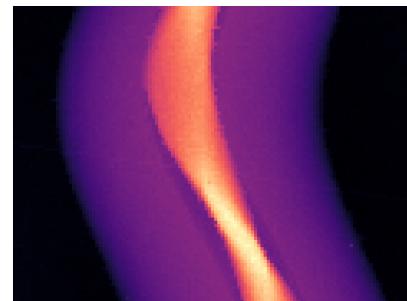


Figura 3.17: Máscara de los datos con artefactos.

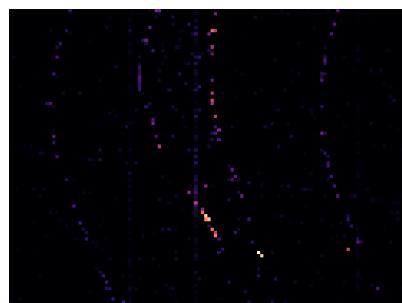
Una vez aplicada la máscara, obtenemos el sinograma mostrado en la Figura 3.18b, en todos los casos, los sinogramas mostrados en la Figura 3.18 son de 90 ángulos y 120 píxeles. Finalmente, al aplicar el algoritmo FBP tanto a los datos originales como co-



(a) Sinograma original.



(b) Sinograma con máscara aplicada.



(c) Diferencia entre sinogramas.

Figura 3.18: Comparativa de sinogramas con y sin máscara.

rregidos, obtenemos la Figura 3.19. Note que la imagen obtenida es similar a la imagen sin artefactos y además significativamente mejor que la obtenida al obviar la corrección realizada en la Figura 3.16b.

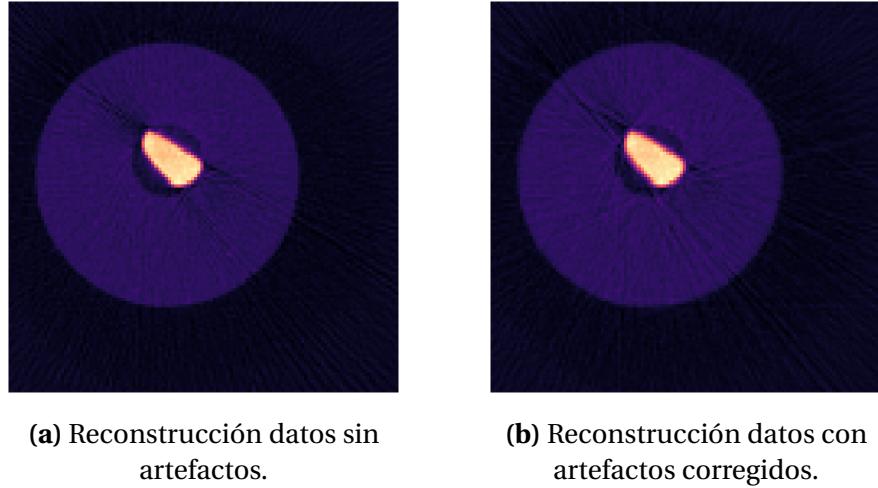


Figura 3.19: Comparativa de reconstrucción FBP con artefactos corregidos.

3.2.4. Reconstrucción de imágenes usando PDHG

En esta sección vamos a revisar la aplicación práctica de la teoría descrita en la sección 2.5.1 para la reconstrucción de imágenes. Además, con el objetivo de visualizar las ventajas que aportan los métodos de regularización vamos a tomar solo una porción de los datos. De forma más precisa, para las secciones en las que abordemos los distintos métodos de regularización, vamos a tomar solo uno de cada 10 sinogramas disponibles en función del ángulo. Para una descripción más detallada de los algoritmos PDHG aplicados en esta sección se recomienda [10].

Consideremos el siguiente problema de minimización:

$$u^* = \arg \min_u \frac{1}{2} \|Au - u^\diamond\|^2 + \begin{cases} \alpha \|u\|_1, \\ \alpha \|\nabla u\|_2^2, \\ \alpha \text{TV}(u) + \mathbb{I}_{\{u \geq 0\}}. \end{cases} \quad (3.1)$$

En el cual, el último término corresponde a las diferentes formas de regularización que vamos a experimentar y donde:

1. u^\diamond es la imagen obtenida del equipo,
2. A es el operador de proyección, en este caso la transformada de Radón, que a nivel de código se define como un operador $A: \mathbb{X} \rightarrow \mathbb{Y}$, donde \mathbb{X} corresponde al objeto `ImageGeometry` y \mathbb{Y} al objeto `AcquisitionGeometry`,
3. α es el parámetro de regularización,
4. TV es la variación total isotrópica definida como

$$\text{TV}(u) = \|\nabla u\|_{2,1},$$

5. $\mathbb{I}_{\{u \geq 0\}}$ está definida por partes de la siguiente manera

$$\mathbb{I}_{\{u \geq 0\}}(u) = \begin{cases} 0 & \text{si } u \geq 0, \\ \infty & \text{en otro caso.} \end{cases}$$

En esta sección vamos a realizar la aplicación de los algoritmos PDHG para la reconstrucción de imágenes con base en el modelo descrito.

Datos

Para la experimentación se utilizaron los datos [D1](#). De manera similar a lo hecho en secciones anteriores, utilizando el código [A.14](#) se obtuvo los datos de la geometría, su representación y proyecciones que mostramos en las Figuras [3.20](#), [3.21b](#); donde el sinograma tiene dimensiones 801×1024 , es decir 801 proyecciones con 1024 píxeles cada una y la proyección es de 1024×1024 píxeles.

```
3D Cone-beam tomography
System configuration:
  Source position: [ 0.          , -295.02502441,   0.          ]
  Rotation axis position: [0., 0., 0.]
  Rotation axis direction: [0., 0., 1.]
  Detector position: [ 0.          , 35.07194901,   0.          ]
  Detector direction x: [1., 0., 0.]
  Detector direction y: [0., 0., 1.]
Panel configuration:
  Number of pixels: [1024 1024]
  Pixel size: [0.06585424 0.06585424]
  Pixel origin: bottom-left
```

```

Channel configuration:
    Number of channels: 1

Acquisition description:
    Number of positions: 801
    Angles 0-20 in radians:
    [-3.1415665, -3.1337643, -3.1259122, -3.118019, -3.1101823,
     -3.1023471, -3.094488, -3.0866292, -3.0787868, -3.070933,
     -3.063082, -3.0552533, -3.0473654, -3.039515, -3.03167,
     -3.023838, -3.0159361, -3.0081236, -3.0002496, -2.992373]

Distances in units: units distance

```

| | | |
|---|--|---|
| — world coordinate system ★ source position | ● rotation axis position — rotation axis direction image geometry × data origin (voxel 0) --- rotation direction θ | ● detector position — detector direction detector × data origin (pixel 0) |
|---|--|---|

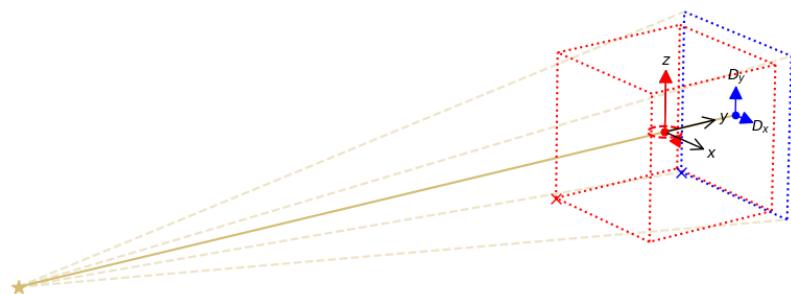
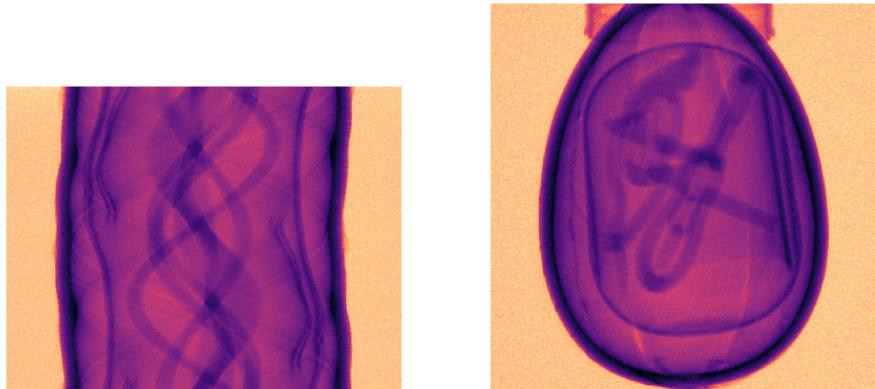


Figura 3.20: Geometría de los datos para PDHG.



(a) Sinograma de los datos para PDHG.

(b) Proyección de los datos para PDHG.

Figura 3.21: Sinograma y proyección de los datos para PDHG.

Reconstrucción con FBP

Una vez cargados los datos vamos a realizar al reconstrucción estándar usando el algoritmo FBP. En la Figura 3.22 se muestra la reconstrucción obtenida con el código A.15 con una resolución de 1024×1024 píxeles y utilizando la totalidad de los datos disponibles. Posteriormente, utilizaremos esta imagen para determinar la calidad de las reconstrucciones obtenidas con los métodos de regularización cuando se toma solo una porción de los datos.



Figura 3.22: Reconstrucción FBP.

Reconstrucción con regularización L_1

Para la aplicación del método PDHG con regularización L_1 primero debemos expresar el modelo (3.1) en términos de la forma estándar descrita en (2.20) con las condiciones ahí mencionadas. Para ello, notemos las siguientes consideraciones utilizando las notaciones de la Sección 2.6:

1. $A = K$ es el operador de proyección:

```
A = ProjectionOperator(ig2D, ag2D, device = "gpu")
```

2. $f(u) = \frac{1}{2} \|u - u^*\|^2$:

```
F = 0.5 * L2NormSquared(b=absorption_data)
```

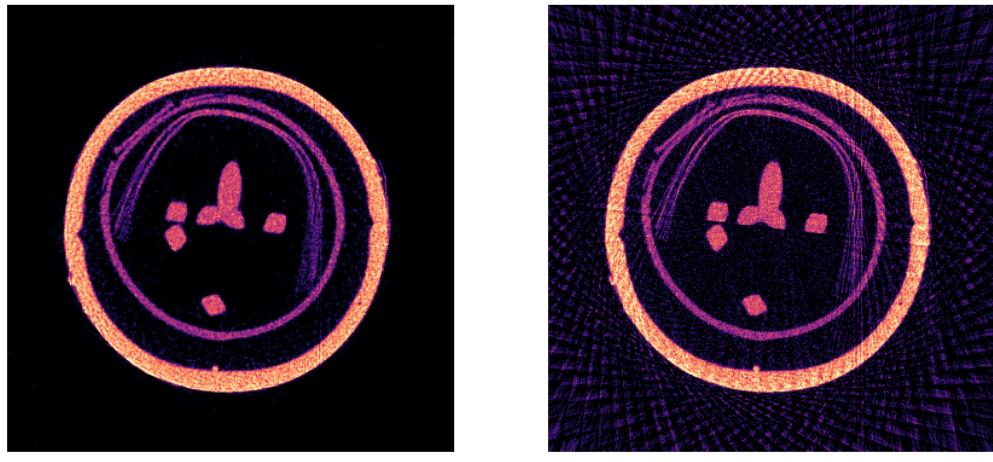
3. $g(u) = \alpha \|u\|_1$:

```

alpha = 0.01
G = alpha * L1Norm()

```

De esta manera, con el código A.16 se obtiene la reconstrucción mostrada en la Figura 3.23; además, se calculan los valores de **SSIM** y **PSNR** para comparar la reconstrucción obtenida con la regularización para datos incompletos, respecto a una reconstrucción ideal con todos los datos utilizando FBP.



(a) Reconstrucción con regularización L_1 .
SSIM: 0,298, **PSNR:** 14,835.

(b) Reconstrucción con FBP.
SSIM: 0,324, **PSNR:** 15,130.

Figura 3.23: Comparativa reconstrucción con regularización L_1 vs FBP.

Reconstrucción con regularización de variación total

De manera similar al caso anterior, para la aplicación del método PDHG con regularización total primero debemos expresar el modelo (3.1) en términos de la forma estándar descrita en (2.20) con las condiciones ahí mencionadas. Para ello, primero definamos las siguientes funciones:

$$\begin{aligned} f_1: \mathbb{Y} &\longrightarrow \mathbb{R} \\ z &\longmapsto \alpha \|z\|_{2,1}, \end{aligned} \tag{3.2}$$

$$\begin{aligned} f_2: \mathbb{X} &\longrightarrow \mathbb{R} \\ z &\longmapsto \frac{1}{2} \|z - u^\diamond\|_2^2, \end{aligned} \tag{3.3}$$

donde la función descrita en (3.2) corresponde al término de regularización de variación total y la función descrita en (3.3) corresponde al término de ajuste de los datos. Entonces, definimos la función F dada por

$$F: \mathbb{Y} \times \mathbb{X} \longrightarrow \mathbb{R}$$

$$(z_1, z_2) \longmapsto f_1(z_1) + f_2(z_2).$$

Lo anterior, se implementa de la siguiente manera:

```
alpha_tv = 0.0003
f1 = alpha_tv * MixedL21Norm()
f2 = 0.5 * L2NormSquared(b=absorption_data)
F = BlockFunction(f1, f2)
```

Por otra parte, es necesario definir otro operador por bloques para incluir el operador gradiente y proyección, es decir, el operador K en la forma estándar. Para ello, definimos el operador K como:

$$K = \begin{pmatrix} \nabla \\ A \end{pmatrix},$$

el cual se implementa de la siguiente manera:

```
K = BlockOperator(Grad, A)
```

Finalmente, para la restricción de no negatividad, definimos la función indicadora de no negatividad como:

$$\mathbb{I}_{\{u \geq 0\}}(u) = \begin{cases} 0 & \text{si } u \geq 0, \\ \infty & \text{en otro caso.} \end{cases}$$

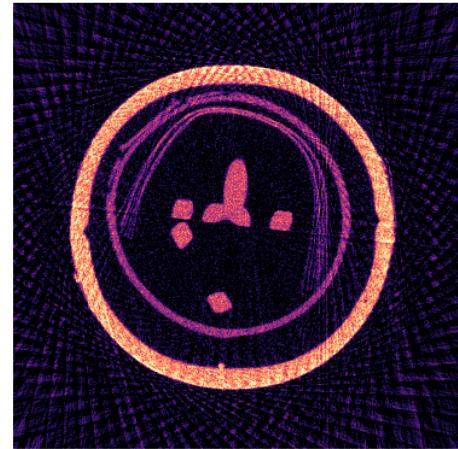
Donde esta función se implementa de la siguiente manera:

```
G = IndicatorBox(lower=0)
```

De esta manera, con el código A.17 se obtiene la reconstrucción mostrada en la Figura 3.24.



(a) Reconstrucción con regularización de variación total.
SSIM: 0,093, **PSNR:** 13,402



(b) Reconstrucción con FBP.
SSIM: 0,324., **PSNR:** 15,130.

Figura 3.24: Reconstrucción con regularización de variación total.

Reconstrucción con regularización de Tykhonov

Este caso corresponde a una ligera modificación del caso anterior en el cual se modifica la definición de la función f_1 en (3.2) por la función:

$$\begin{aligned} f_1: \mathbb{Y} &\longrightarrow \mathbb{R} \\ z &\longmapsto \alpha \|z\|_2^2. \end{aligned} \tag{3.4}$$

Lo anterior, se implementa de la siguiente manera:

```
alpha_tykhonov = 0.0003
f1 = alpha_tykhonov * L2NormSquared()
```

De esta manera, con el código A.18 se obtiene la reconstrucción mostrada en la Figura 3.25.

Finalmente, en la Tabla 3.4 se muestran los tiempos de ejecución de los diferentes métodos de regularización. Mientras que, en la Figura 3.26 se muestran las reconstrucciones obtenidas con los diferentes métodos de regularización.



(a) Reconstrucción con regularización de Tikhonov.
SSIM: 0,104, **PSNR:** 13,569.



(b) Reconstrucción con FBP.
SSIM: 0,324, **PSNR:** 15,130.

Figura 3.25: Reconstrucción con regularización de Tykhonov.

| Método | Tiempo de ejecución (s) |
|----------|-------------------------|
| L_1 | 10.5 |
| TV | 38.5 |
| Tykhonov | 29.9 |

Tabla 3.4: Comparación de tiempos de ejecución de los métodos de regularización.

3.3. NiftyPET

En la práctica, las imágenes por emisión de positrones se utilizan de manera conjunta con otros estudios de imagenología para obtener información detallada sobre el órgano o la región de estudio. Recordemos que las imágenes PET corresponden a un tipo funcional de imagen mientras que las imágenes CT, MR, etc. corresponden a un tipo anatómico. En esta sección vamos a revisar el uso de la librería NiftyPET para la reconstrucción de imágenes PET.

3.3.1. Carga de los datos

Para el desarrollo de los siguientes experimentos se utilizó el conjunto de datos [D2](#), estos incluyen:



(a) FBP datos completos.



(b) Regularización L_1 .



(c) Regularización de variación total.



(d) Regularización de Tykhonov.

Figura 3.26: Reconstrucciones con PDHG usando diferentes métodos de regularización.

1. Datos PET en modo lista de 60 minutos de adquisición,
2. Datos de atenuación (mapa de atenuación o μ -map),
3. Datos de normalización.

Los datos mencionados, en sus respectivos formatos, deben estar en una carpeta con la siguiente estructura:

De esta manera, la función `classify_input` de NiftyPET nos permite cargar los datos en un diccionario que lo identifica con su respectiva componente. En el código A.19 se muestra cómo se cargan los datos. De esta manera, se obtiene la siguiente salida:

```
INFO:niftypet.nipet.mmraux:got LM interfile.
```

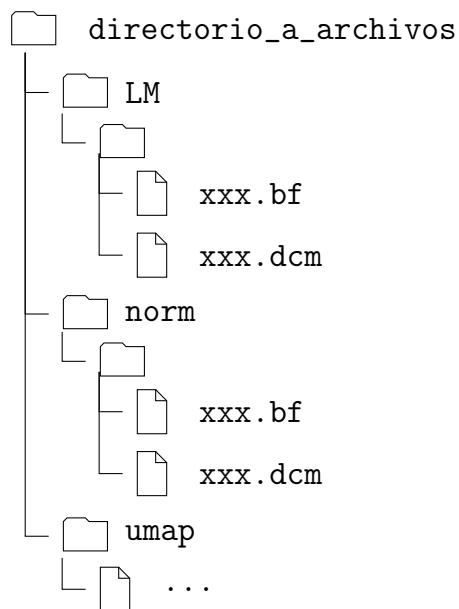


Figura 3.27: Estructura de directorios para los datos de PET.

```

INFO:niftypet.nipet.mmrax:got CSA info.

{
  'corepath': '/mnt/test1dev/TIC/00Datos/amyloidPET_FBP_TP0',
  'lm_dcm': '/mnt/test1dev/TIC/00Datos/amyloidPET_FBP_TP0/LM/17598013
             _1946_20150604155500.000000.dcm',
  'lm_bf': '/mnt/test1dev/TIC/00Datos/amyloidPET_FBP_TP0/LM/17598013
             _1946_20150604155500.000000.bf',
  'nrm_dcm': '/mnt/test1dev/TIC/00Datos/amyloidPET_FBP_TP0/norm
              /17598013_1946_20150604082431.000000.dcm',
  'nrm_bf': '/mnt/test1dev/TIC/00Datos/amyloidPET_FBP_TP0/norm/17598013
              _1946_20150604082431.000000.bf',
  'mumapDCM': '/mnt/test1dev/TIC/00Datos/amyloidPET_FBP_TP0/umap',
  '#mumapDCM': 192
}

```

3.3.2. Mapas de atenuación

Dado que la imagen PET corresponde a una medición de la radiación emitida por el radiofármaco, es necesario conocer la forma en la que atenúa el medio (paciente y

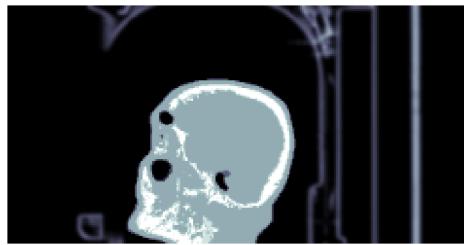
equipo) a la radiación emitida por el medicamento utilizado. En este sentido, el conjunto de datos con el que estamos trabajando fue generado a partir de un escáner **Siemens Biograph mMR** el cual es un escáner PET-MR, dado que a partir de una resonancia magnética no es posible obtener la información de atenuación, estos datos deben ser generados a partir de una tomografía computarizada y usualmente son proporcionados por el fabricante del equipo.

En el código A.20 se muestra la extracción de los mapas de atenuación para el equipo y el paciente basado en la MR obtenida con el mismo escáner. En la Figura 3.28 se muestran los mapas de atenuación para el equipo y el paciente, respectivamente, en ambos casos las dimensiones de los mapas son de $344 \times 344 \times 127$, píxeles. Por otro lado, la Figura 3.29 se muestra un ejemplo de las diferentes vistas que se pueden obtener.

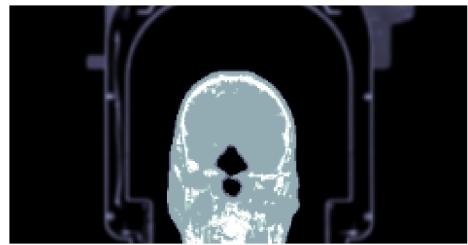


Figura 3.28: Mapas de atenuación con corte axial.

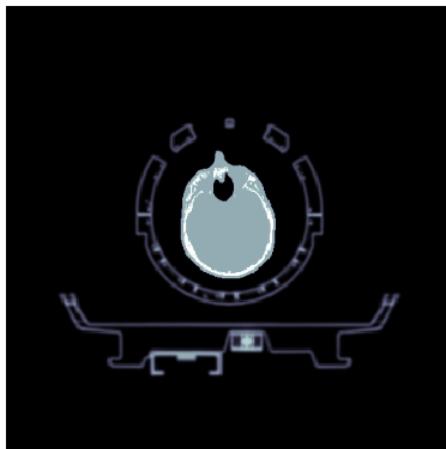
Dado que los mapas de atenuación son en 3 dimensiones, es posible obtener vistas de los mismos para diferentes tipos de planos. Primero, en la Figura 3.29c se muestra un ejemplo para un plano axial con dimensiones de 344×344 píxeles. Por otro lado, en la Figura 3.29b se muestra un ejemplo para un plano coronal con dimensiones de 127×244 píxeles (el número de píxeles es menor debido a un recorte aplicado para mejorar la visualización, el mapa completo tiene dimensiones de 127×344 píxeles). Finalmente, en la Figura 3.29a se muestra un ejemplo para un plano sagital con dimensiones de 127×244 debido al mismo motivo descrito anteriormente.



(a) Mapa de atenuación corte sagital.



(b) Mapa de atenuación corte coronal.



(c) Mapa de atenuación corte axial.

Figura 3.29: Mapas de atenuación con diferentes cortes.

3.3.3. Sinogramas

Una vez cargados los datos y los mapas de atenuación, el paso previo a la reconstrucción es visualizar los sinogramas obtenidos. Recordemos que para el caso PET consideramos los siguientes:

1. **Sinograma**¹: contienen todos los eventos detectados por el equipo,
2. **Sinograma retardado**²: contienen los eventos detectados después de un tiempo de espera, de modo que buscan registrar únicamente coincidencias aleatorias.

En la Figura 3.30 se muestran ejemplos de los sinogramas obtenidos. Mientras que en la Figura 3.31 se muestra el conteo de los eventos detectados en cada sinograma a lo

¹En inglés: *prompt sinogram*

²En inglés: *delayed sinogram*

largo del estudio. El código para la obtención de los sinogramas se muestra en el código A.21 y las dimensiones en ambos casos son: 252×344 , píxeles.

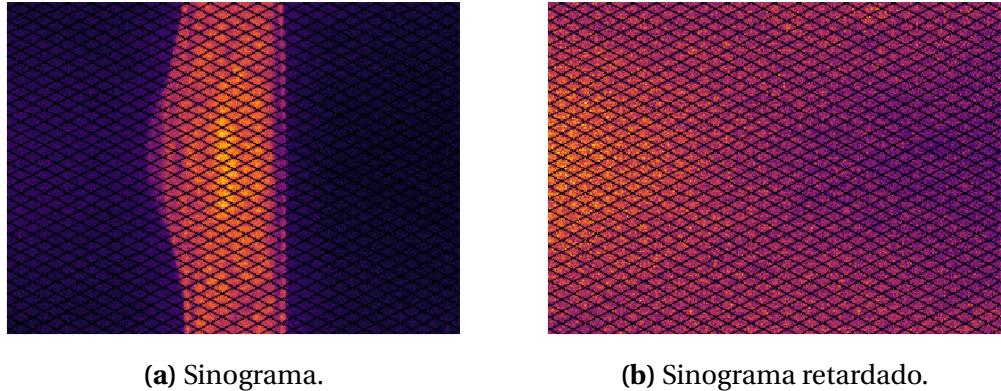


Figura 3.30: Sinogramas PET.

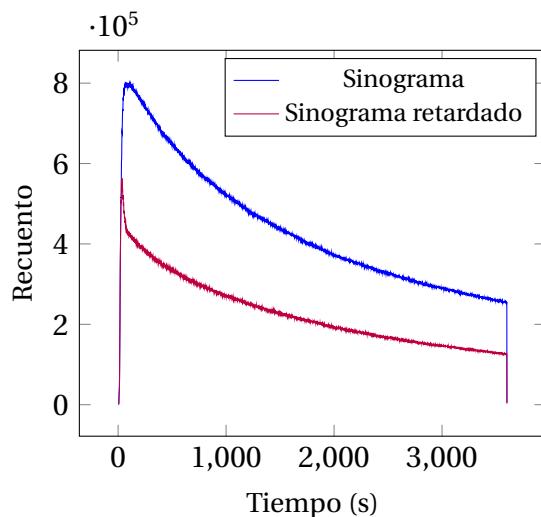


Figura 3.31: Conteo de eventos en los sinogramas.

3.3.4. Reconstrucción

Para la reconstrucción de la imagen PET, realizaremos tres etapas: En la primera utilizaremos el algoritmo ML-EM, el cual conocemos experimentalmente converge aproximadamente al resultado buscado, pero es lento. Así, realizaremos la reconstrucción con 1000 iteraciones de las cuales registraremos la imagen obtenida en las iteraciones múltiplos de 10. Con esta información realizaremos una exploración visual de la evolución de la reconstrucción con la finalidad de determinar empíricamente el número

de iteraciones necesarias para obtener una reconstrucción aceptable. Para el segundo paso, utilizaremos la métrica SSIM aplicada a 3 cortes en los tres planos (axial, coronal y sagital) para evaluar la calidad de la reconstrucción y detener la ejecución del algoritmo cuando el promedio de las métricas sea menor a la tolerancia que asegure que la reconstrucción pare en el número de iteraciones determinado en la primera etapa. Para la última etapa, utilizaremos el algoritmo OSEM y tomando como criterio de parada la métrica estimada entre la última iteración de ML-EM y la iteración actual de OSEM. Finalmente, compararemos los resultados obtenidos en términos de número de iteraciones, tiempo de ejecución y calidad de la reconstrucción obtenidos en los pasos dos y tres.

Reconstrucción: ML-EM sin criterio de parada

Debido a la cantidad de resultados, estos se encuentran disponibles para su visualización en la carpeta [reconstrucion_inicial](#) del repositorio. Así, visualmente se determinó que a partir de la iteración 300, aproximadamente, existe un deterioro en la imagen y una imagen de mejor calidad se obtiene al rededor de la iteración 200. En la Figura 3.32 se muestra la evolución del promedio de la métrica SSIM de los cortes axial, coronal y sagital en función del número de iteraciones. Nuevamente, se muestra una mejora continua significativa hasta, aproximadamente, la iteración 200. A partir de esta la mejora es lenta y por el contrario, como se puede observar en las imágenes, la calidad de la reconstrucción disminuye.

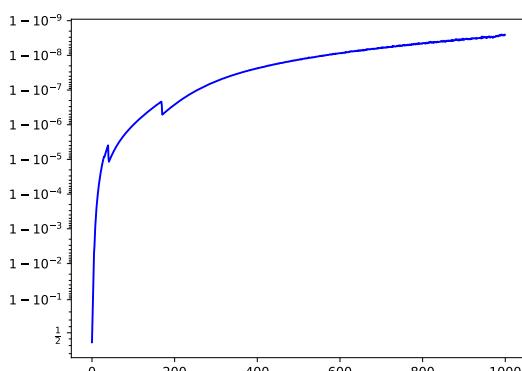


Figura 3.32: Evolución de la métrica SSIM en función del número de iteraciones.

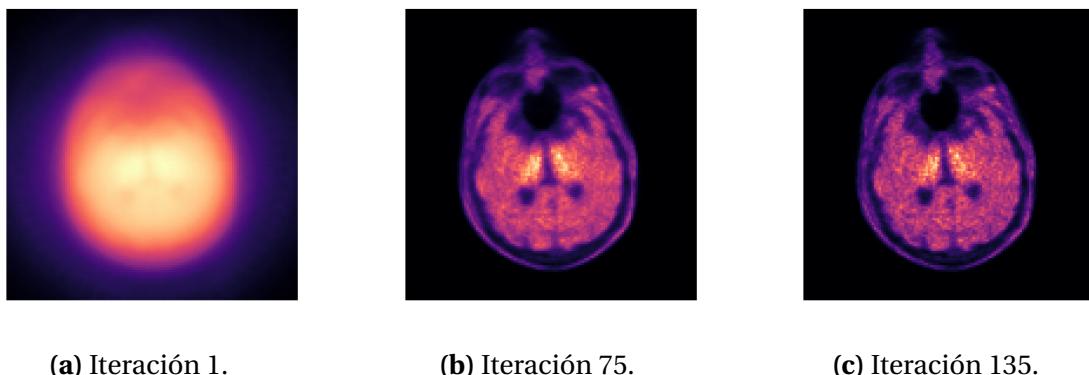
De esta manera, y realizando varias ejecuciones se determinó que una tolerancia de

1×10^{-8} define un parámetro de parada adecuado que permite obtener una reconstrucción aceptable con un buen nivel de detalle. En la siguiente secciones presentaremos los resultados obtenidos para la reconstrucción realizada con la tolerancia mencionada.

Reconstrucción: ML-EM con criterio de parada

Con el criterio de parada establecido, en las Figuras 3.33, 3.34 y 3.35 se muestran los resultados para los cortes axial, sagital y coronal, respectivamente, en algunas iteraciones lo que muestra el progreso del algoritmo de reconstrucción. Las dimensiones para los cortes axiales son: 480×480 , para los cortes son 489×480 y para los cortes coronales son 595×480 , píxeles.

Las gráficas de cada iteración se encuentran disponibles en la siguiente carpeta: [reconstrucción_mlem](#).



(a) Iteración 1.

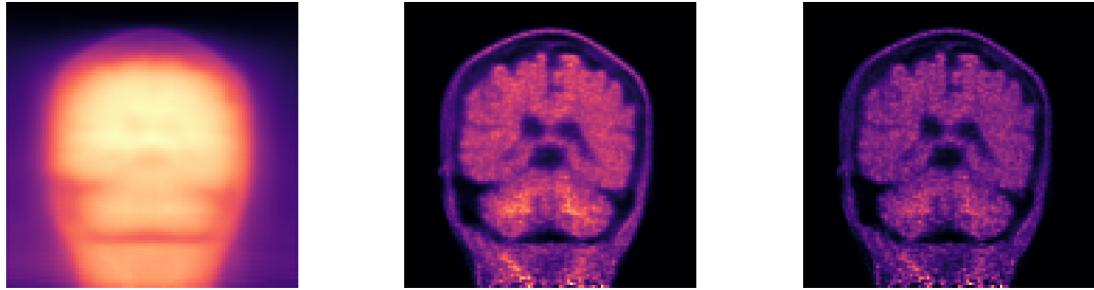
(b) Iteración 75.

(c) Iteración 135.

Figura 3.33: Reconstrucción ML-EM corte axial.

La Tabla 3.5 muestra los tiempos aproximados de ejecución para los diferentes pasos realizados excepto la reconstrucción que depende del número de iteraciones, es importante notar que NiftyPET permite almacenar varios datos utilizados en la reconstrucción lo que evita su generación en cada ejecución del código.

Por otra parte, dado que los tiempos entre iteración son aproximadamente similares (en promedio 5,13 segundos) y los resultados dependen del número de iteraciones, en la Figura 3.36, se ilustra la relación entre el número de iteraciones y el tiempo de ejecución acumulado hasta la iteración correspondiente. Note que dado el tiempo casi constante, la relación es lineal por lo que se puede estimar el tiempo total de ejecución para un

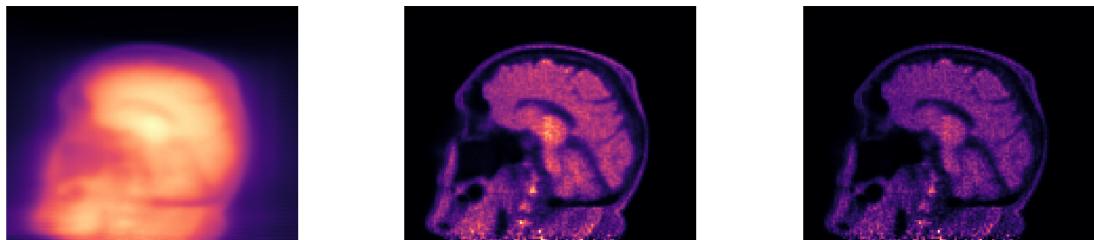


(a) Iteración 1.

(b) Iteración 75.

(c) Iteración 135.

Figura 3.34: Reconstrucción ML-EM corte sagital.



(a) Iteración 1.

(b) Iteración 75.

(c) Iteración 135.

Figura 3.35: Reconstrucción ML-EM con 50 iteraciones.

| Paso | Tiempo aproximado (s) |
|---|-----------------------|
| Adquisición de datos | 15 |
| Mapas de atenuación | 20 |
| Preprocesamiento | 40 |
| Preparación de los datos en formato lista | 160 |

Tabla 3.5: Tiempos de ejecución.

número de iteraciones dado.

Por otra parte, dado que se utilizó la métrica SSIM para evaluar la calidad de la reconstrucción, en la Figura 3.37 se muestra el valor de la métrica para diferentes iteraciones. Note que el valor de la métrica aumenta con el número de iteraciones, lo cual es esperado dado que el algoritmo ML-EM converge al resultado buscado. Además, note que la convergencia es significativamente lenta, por lo que se requieren un número considerable de iteraciones para obtener una reconstrucción aceptable en términos de

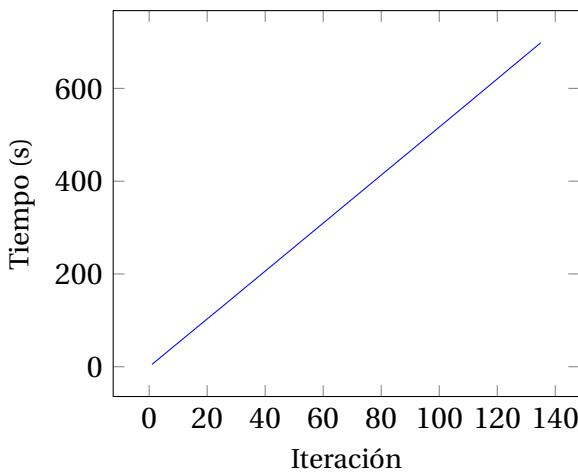


Figura 3.36: Tiempos de ejecución acumulados por iteración ML-EM.

la métrica SSIM aunque, dada la tolerancia establecida, el tiempo de ejecución es aceptable.

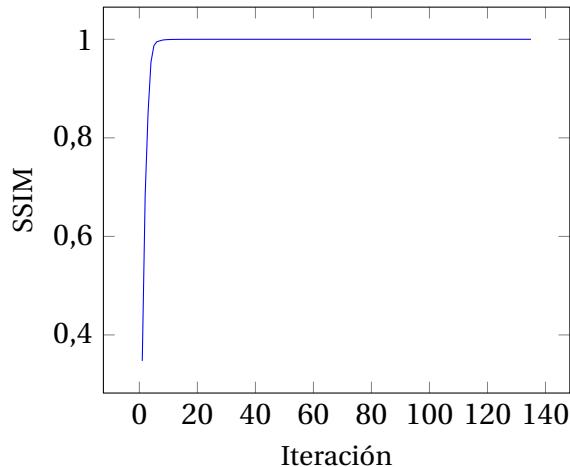


Figura 3.37: SSIM por iteración ML-EM.

Finalmente, la Tabla A.1 disponible en el Anexo A.4 incluye todos los parámetros calculados para la reconstrucción con ML-EM.

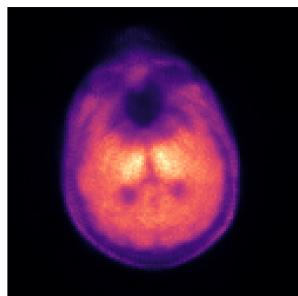
OSEM

Una vez hemos obtenido una imagen³ de referencia con el algoritmo ML-EM, vamos a comprobar de manera experimental si el algoritmo OSEM converge a un resultado si-

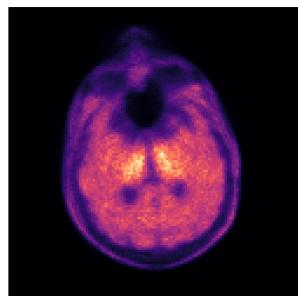
³Ver Figuras 3.34c, 3.35c y 3.33c

milar en un tiempo menor. Para este fin, utilizaremos la misma metodología que en la sección anterior, es decir, utilizaremos la métrica SSIM para evaluar la calidad de la reconstrucción. No obstante, como el método OSEM puede sobreajustarse al ruido de los datos, la tolerancia de convergencia elegida debe ser cuidadosamente seleccionada para evitar este problema. En este sentido, luego de varias pruebas realizadas, se determinó que una tolerancia de 0,005 es adecuada para evitar el sobreajuste mencionado.

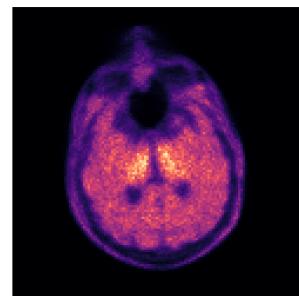
Una vez ejecutada la reconstrucción, en las Figuras 3.38, 3.39 y 3.40 se muestran los resultados obtenidos para los cortes axial, sagital y coronal, respectivamente, en las iteraciones 1, 3 y 5. Las dimensiones para los cortes axiales son: 480×480 , para los cortes son 489×480 y para los cortes coronales son 595×480 , píxeles. Finalmente, las gráficas de cada iteración se encuentran disponibles en la siguiente carpeta: [reconstruccion_osem](#).



(a) Iteración 1.

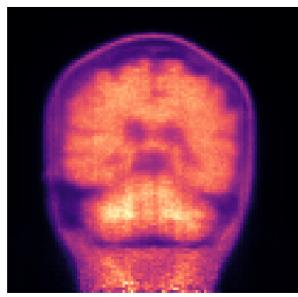


(b) Iteración 3.

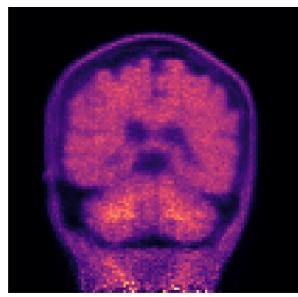


(c) Iteración 5.

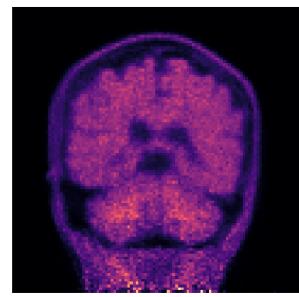
Figura 3.38: Reconstrucción OSEM corte axial.



(a) Iteración 1.

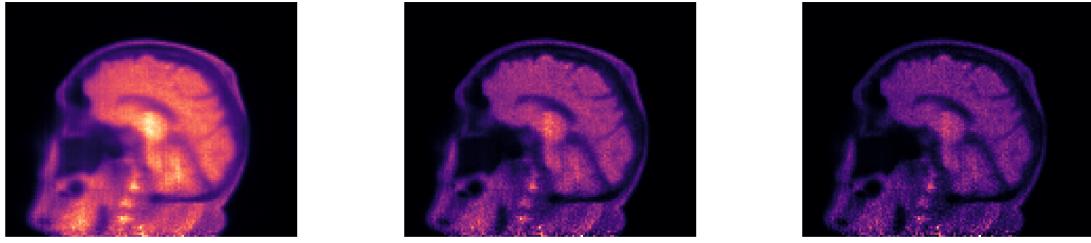


(b) Iteración 3.



(c) Iteración 5.

Figura 3.39: Reconstrucción OSEM corte sagital.



(a) Iteración 1.

(b) Iteración 3.

(c) Iteración 5.

Figura 3.40: Reconstrucción OSEM corte coronal.

Es evidente que en una cantidad significativamente menor de iteraciones se obtiene una reconstrucción aceptable a simple vista. No obstante, las iteraciones de OSEM incrementan su duración lo que se traduce en un mayor tiempo de ejecución. En la Figura 3.41 se muestran los tiempos acumulados para las diferentes iteraciones realizadas. Note que dado que los procesos previos son similares, los datos de la Tabla 3.5 son válidos para este caso.

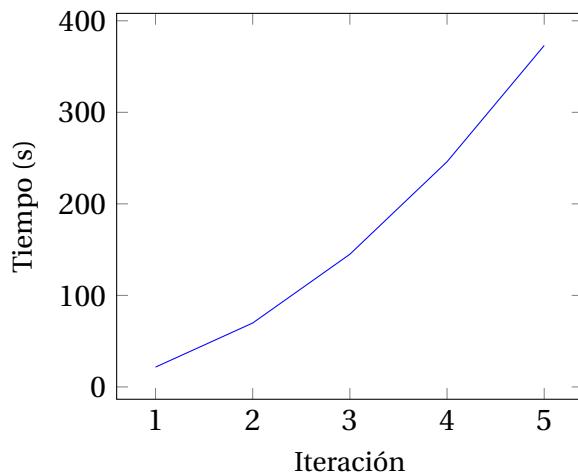


Figura 3.41: Tiempos de ejecución acumulados por iteración OSEM.

Por otro lado, para estudiar la «velocidad» a la que converge el algoritmo OSEM, en la Figura 3.42 se muestra el valor de la métrica SSIM para las diferentes iteraciones. Note que el valor de la métrica aumenta rápidamente en las primeras iteraciones pero parece disminuir para las últimas iteraciones. Esto nos sugiere que el método OSEM converge rápidamente a un resultado aceptable pero luego se estanca y aún más la calidad de las imágenes se deteriora. Otra forma de visualizar este comportamiento es en la Figura 3.43

donde se muestra el «porcentaje» de diferencia de la imagen reconstruida con la imagen de referencia, respecto a SSIM. Note que para la primera iteración ya se obtiene una diferencia de únicamente un 4 % respecto a la imagen de referencia, mientras que para la última iteración la diferencia es cercana al 0,2 %.

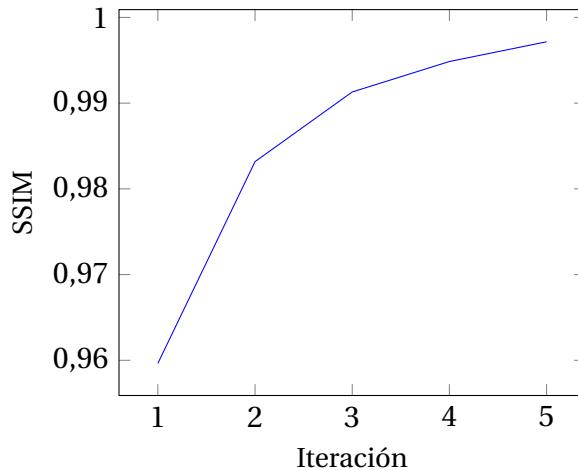


Figura 3.42: Métrica SSIM por iteración OSEM.

Así, gracias a la información mostrada en las Figuras 3.41 y 3.43 podemos concluir que el algoritmo OSEM converge rápidamente a un resultado aceptable (una diferencia menor al 1 % se alcanza en la tercera iteración) para luego estancarse y de hecho incrementar significativamente el tiempo de ejecución sin mejorar la calidad de la imagen.

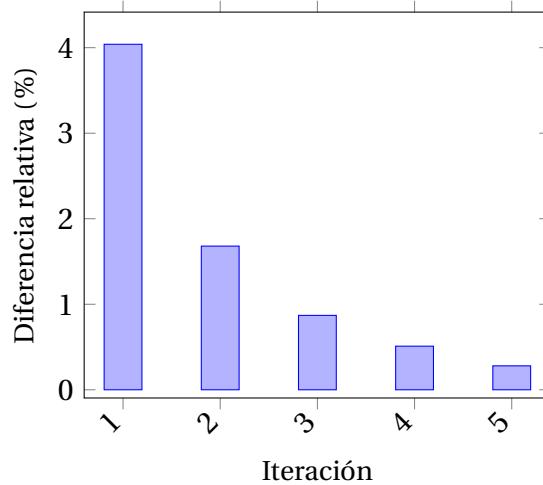


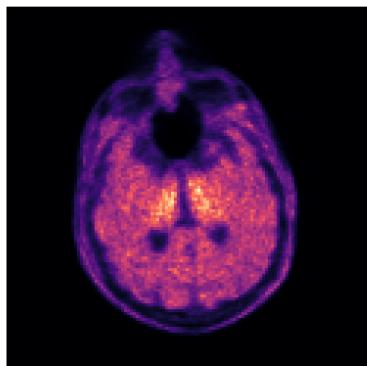
Figura 3.43: Diferencia relativa respecto a la imagen de referencia por iteración OSEM con la métrica SSIM.

Finalmente, la Tabla 3.6 muestra los valores obtenidos en detalle para las métricas

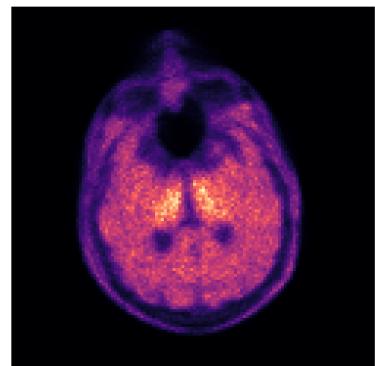
calculadas en reconstrucción OSEM. Mientras que la Figura 3.44 incluye una comparación de los resultados de la métrica SSIM obtenidos con ML-EM y OSEM para la última iteración del método ML-EM (imagen objetivo) y la quinta iteración de OSEM y fue obtenida en aproximadamente 373 segundos mientras que la imagen objetivo fue obtenida en aproximadamente 700 segundos.

| Iteración | Tiempo (s) | PSNR | SSIM | Acumulado (s) | Dif. SSIM |
|-----------|------------|--------|-------|---------------|-----------|
| 1 | 21,645 | 46,543 | 0,960 | 21,645 | 4,040 |
| 2 | 48,172 | 54,298 | 0,983 | 69,817 | 1,680 |
| 3 | 75,121 | 58,755 | 0,991 | 144,938 | 0,870 |
| 4 | 101,085 | 61,187 | 0,995 | 246,024 | 0,510 |
| 5 | 126,957 | 63,446 | 0,997 | 372,980 | 0,280 |

Tabla 3.6: Detalles de la reconstrucción OSEM.

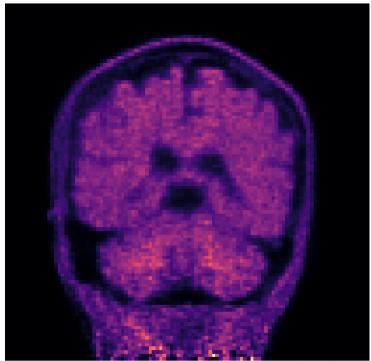


(a) Corte axial ML-EM.

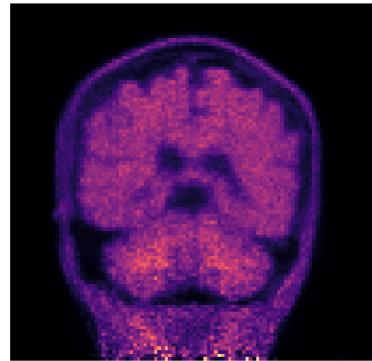


(b) Corte axial OSEM.

Figura 3.44: Comparativa de resultados ML-EM y OSEM.

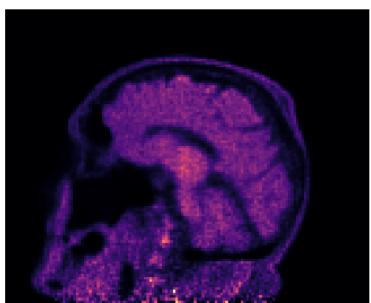


(c) Corte sagital ML-EM.

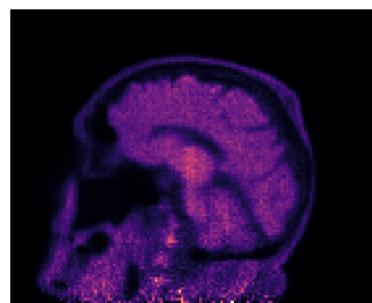


(d) Corte sagital OSEM.

Figura 3.44: Comparativa de resultados ML-EM y OSEM.



(e) Corte sagital ML-EM.



(f) Corte sagital OSEM.

Figura 3.44: Comparativa de resultados ML-EM y OSEM.

Capítulo 4

Conclusiones y Recomendaciones

4.1. Conclusiones

1. La aplicación correcta de la teoría matemática permite utilizar los resultados disponibles en la literatura para avanzar en áreas específicas. En particular, la teoría de la transformada de Fourier y Radón en espacios de Schwartz facilita el empleo de teoremas de inversión y propiedades de los operadores.
2. La teoría de regularización en problemas de optimización permite obtener resultados que se ajustan a las necesidades y condiciones específicas de cada caso. Esto es crucial para abordar problemas de reconstrucción reales, en los que diversos fenómenos físicos y condiciones de adquisición de datos pueden influir en la estabilidad y calidad de las reconstrucciones realizadas con métodos analíticos.
3. Dada la complejidad del objeto de estudio (una imagen), seleccionar una métrica adecuada para evaluar la calidad de la reconstrucción es esencial. Utilizar métricas especializadas como PSNR y SSIM ofrece una mejora significativa en los resultados en comparación con métricas no especializadas, como el error cuadrático medio.
4. La amplia variedad de tipos de regularización disponibles para los problemas de optimización hace crucial clasificar su uso según las necesidades y limitaciones prácticas. Como se muestra en la Tabla 3.4, cada tipo de regularización tiene un

coste computacional diferente, y en algunos casos, considerable.

5. El uso de diversas técnicas para la obtención de imágenes con fines médicos facilita la detección y tratamiento de enfermedades que, de otro modo, podrían pasar desapercibidas para los especialistas. Como se detalló en la sección [2.7.5](#), las nuevas tecnologías y mejoras en la reconstrucción de imágenes proporcionan herramientas más efectivas para el diagnóstico y tratamiento de enfermedades.

4.2. Recomendaciones

1. Debido a la limitada presencia de escáneres tipo PET en el país y a la alta especialización de cada fabricante, se recomienda llevar a cabo estudios orientados a los equipos disponibles localmente. Esto facilitaría la obtención de conjuntos de datos necesarios para la experimentación.
2. Para una comprensión adecuada de los conceptos tanto matemáticos como médicos, se recomienda comenzar con el estudio de los rayos X, seguido de la tomografía computarizada y, finalmente, la tomografía por emisión de positrones. Este enfoque facilita la comprensión de la finalidad de cada técnica y de las diferentes necesidades que justifican el uso combinado de estas tecnologías.
3. Dado que los datos a estudiar son considerablemente grandes (del orden de gigabytes) y las operaciones son intensivas, se recomienda utilizar equipos con alta capacidad de memoria, procesamiento y acceso a tarjetas gráficas para acelerar los cálculos. En el transcurso de este trabajo, el uso de recursos adecuados redujo significativamente los tiempos de ensayo.
4. Para ejecutar experimentos con un gran número de iteraciones, se recomienda optimizar la gestión de los datos almacenados en cada iteración para evitar la saturación de la memoria RAM. Se puede optar por separar la experimentación en etapas o guardar los datos cada cierto número de iteraciones, lo que permite liberar espacio en la memoria y continuar con los ensayos sin interrupciones.

Capítulo A

Anexos

A.1. Notación

Símbolo **Significado**

| | |
|---------------------|---|
| \mathbb{C}^n | Cilindro unitario en \mathbb{R}^n . |
| $D(\Omega)$ | Espacio de funciones suaves con soporte compacto. |
| $D(\Omega)'$ | Espacio de distribuciones. |
| \mathcal{F} | Transformada de Fourier. |
| \tilde{f} | Reflexión de f . |
| $H(\theta, s)$ | Hiperplano en \mathbb{R}^n . |
| $L^p(\mathbb{R}^n)$ | Espacio de funciones p integrables en \mathbb{R}^n . |
| \mathbb{N}_0 | Conjunto de los números naturales incluyendo al cero. |
| \mathbb{N}_0^n | Producto cartesiano de \mathbb{N}_0 consigo mismo n veces. (Conjunto de multi-índices). |
| \mathcal{S} | Espacio de Schwartz (funciones a decaimiento rápido). |
| S_n | Esfera unitaria en \mathbb{R}^n . |
| R | Transformada de Radón. |
| $*$ | Operador convolución. |
| δ | Función delta de Dirac. |
| τ^y | Traslación en \mathbb{R}^n . |
| δ^a | Dilación en \mathbb{R}^n . |

A.2. Códigos

En cada uno de los siguientes scripts se utilizan los siguientes módulos:

```
1 from PIL import Image  
2 import numpy as np  
3 import pandas as pd  
4 import matplotlib.pyplot as plt
```

Código A.1: Código la obtención de la matriz en escala de grises

```
1 # Ruta a la imagen JPG  
2 image_path = "ImagenBase.png"  
3  
4 # Cargar la imagen  
5 image = Image.open(image_path)  
6  
7 # Convertir la imagen a escala de grises (opcional)  
8 image_gray = image.convert("L")  
9 data_matrix_gray = np.array(image_gray)  
10  
11 # get in latex form for easy copy-paste  
12 pd.DataFrame(data_matrix_gray).to_latex(index=False, header=False,  
    escape=False)
```

Código A.2: Código la obtención del histograma

```
1 # Ruta a la imagen JPG  
2 image_path = "Cotopaxi.jpg"  
3  
4 # Cargar la imagen  
5 image = Image.open(image_path)  
6  
7 # Convertir la imagen a escala de grises (opcional)  
8 image_gray = image.convert("L")  
9  
10 # Guardar la imagen en escala de grises  
11 image_gray.save("Cotopaxi_gray.jpg")
```

```

12
13 data_matrix_gray = np.array(image_gray)
14
15 # get in latex form for easy copy-paste
16 pd.DataFrame(data_matrix_gray).to_latex(index=False, header=False,
   escape=False)
17
18 # Calcular el histograma
19 histogram, bins = np.histogram(data_matrix_gray, bins=256, range=(0,
   255))
20
21 # Graficar el histograma
22 plt.figure()
23 plt.bar(bins[:-1], histogram, width=1, color='gray')
24 plt.title("Histograma de la Imagen en Escala de Grises")
25 plt.xlabel("Intensidad de los Pixelles")
26 plt.ylabel("Frecuencia")
27 plt.show()
28
29 # Guardar el histograma en un archivo
30 output_file = "histogram_data.txt"
31 np.savetxt(output_file, histogram, fmt="%d")

```

Código A.3: Código la obtención de la función de distribución acumulada

```

1 # Obtenemos la gráfica de la función de distribución acumulada
2 cdf = np.cumsum(histogram)
3
4 # Graficar la función de distribución acumulada
5 plt.figure()
6 plt.plot(bins[:-1], cdf, color='gray')
7 plt.title("Función de Distribución Acumulada")
8 plt.xlabel("Intensidad de los Pixelles")
9 plt.ylabel("Frecuencia Acumulada")
10 plt.show()

```

Código A.4: Código la obtención de la imagen ecualizada

```

1 # Calcular la función de distribución acumulada normalizada
2 cdf_normalized = cdf / cdf[-1]
3
4 # Crear un diccionario para mapear los valores de intensidad de los
   pixeles
5
6 mapping = {}
7
8 for i in range(256):
9     mapping[i] = np.round(255 * cdf_normalized[i])
10
11 # Crear una nueva imagen con el histograma ecualizado
12 data_matrix_eq = np.zeros_like(data_matrix_gray)
13
14 for i in range(data_matrix_gray.shape[0]):
15
16     for j in range(data_matrix_gray.shape[1]):
17         data_matrix_eq[i, j] = mapping[data_matrix_gray[i, j]]
18
19 # Convertir la matriz de datos a un objeto de imagen
20 image_eq = Image.fromarray(data_matrix_eq)
21
22 # Guardar la imagen ecualizada
23 image_eq.save("Cotopaxi_eq.jpg")
24
25 # Hacemos un gráfico del histograma de la imagen ecualizada
26 histogram_eq, bins_eq = np.histogram(data_matrix_eq, bins=256, range=(0,
   255))
27
28 plt.figure()
29 plt.bar(bins_eq[:-1], histogram_eq, width=1, color='gray')
30 plt.title("Histograma de la Imagen Ecualizada")
31 plt.xlabel("Intensidad de los Pixeles")
32 plt.ylabel("Frecuencia")
33 plt.show()
34

```

```

35
36 # Graficamos la función de distribución acumulada de la imagen
37 # ecualizada
38
39 plt.figure()
40 plt.plot(bins_eq[:-1], cdf_eq, color='gray')
41 plt.title("Función de Distribución Acumulada de la Imagen Ecualizada")
42 plt.xlabel("Intensidad de los Pixelés")
43 plt.ylabel("Frecuencia Acumulada")
44 plt.show()

```

Código A.5: Código la obtención de primera derivada de la imagen.

```

1 # Calculamos la primera derivada en x de la imagen original mediante
2 # diferencias finitas hacia adelante
3 dx = 1
4 data_matrix_dx = np.zeros_like(data_matrix_gray)
5
6 for i in range(data_matrix_gray.shape[0]):
7     for j in range(data_matrix_gray.shape[1] - 1):
8         data_matrix_dx[i, j] = (data_matrix_gray[i, j + 1] -
9             data_matrix_gray[i, j]) / dx
10
11 # Calculamos la primera derivada en y de la imagen original mediante
12 # diferencias finitas hacia adelante
13 dy = 1
14 data_matrix_dy = np.zeros_like(data_matrix_gray)
15
16 for i in range(data_matrix_gray.shape[0] - 1):
17     for j in range(data_matrix_gray.shape[1]):
18         data_matrix_dy[i, j] = (data_matrix_gray[i + 1, j] -
19             data_matrix_gray[i, j]) / dy
20
21 # Mostramos la imagen original y sus derivadas
22 plt.figure()
23 plt.subplot(1, 3, 1)

```

```

20 plt.imshow(data_matrix_gray, cmap='gray')
21 plt.title("Imagen Original")
22 plt.axis("off")
23
24 plt.subplot(1, 3, 2)
25 plt.imshow(data_matrix_dx, cmap='gray')
26 plt.title("Primera Derivada en x")
27 plt.axis("off")
28
29 plt.subplot(1, 3, 3)
30 plt.imshow(data_matrix_dy, cmap='gray')
31 plt.title("Primera Derivada en y")
32 plt.axis("off")
33
34 plt.show()
35
36 # Guardamos las imágenes
37 image_dx = Image.fromarray(data_matrix_dx)
38 image_dx.save("Cotopaxi_dx.jpg")
39
40 image_dy = Image.fromarray(data_matrix_dy)
41 image_dy.save("Cotopaxi_dy.jpg")

```

Código A.6: Eliminación de ruido de una imagen con el modelo ROF

```

1
2 # Importamos las librerías necesarias.
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from skimage import data
6 from skimage.restoration import denoise_tv_chambolle
7 from skimage.util import random_noise
8
9 # Cargar imagen de ejemplo
10 imagen = data.camera()
11
12 # Añadir ruido a la imagen

```

```

13 imagen_ruidosa = random_noise(imagen, var=0.01)
14
15 # Realizar la eliminación de ruido de TV
16 imagen_denoised = denoise_tv_chambolle(imagen_ruidosa, weight=0.1)
17
18 # Mostrar resultados
19 fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(15, 5),
20                         sharex=True, sharey=True)
21 ax[0].imshow(imagen, cmap='gray')
22 ax[0].set_title('Imagen Original')
23 ax[1].imshow(imagen_ruidosa, cmap='gray')
24 ax[1].set_title('Imagen Ruidosa')
25 ax[2].imshow(imagen_denoised, cmap='gray')
26 ax[2].set_title('Imagen Sin Ruido')
27
28 for a in ax:
29     a.axis('off')
30
31 plt.tight_layout()
32 plt.show()

```

Referencia sobre la función utilizada para eliminar el ruido se puede encontrar en: [Módulo: denoise_tv_chambolle](#). Además, es importante resaltar que esta función es una implementación del algoritmo presentado en [9].

Código A.7: Restauración de una imagen borrosa

```

1
2 # Importamos las librerías necesarias.
3 import scico.numpy as snp
4 import scico.random
5 from scico import functional, linop, loss, metric, plot
6 from scico.optimize import ProximalADMM
7 from scico.util import device_info
8 plot.config_notebook_plotting()
9 from skimage import data, color
10 imagen = color.rgb2gray(data.astronaut())
11

```

```

12 x_gt = imagen
13
14 n = 25 # convolution kernel size
15 sigma_p = 0.01 # noise level
16
17 psf = snp.ones((n, n)) / (n * n)
18 C = linop.Convolve(h=psf, input_shape=x_gt.shape)
19
20 Cx = C(x_gt) # blurred image
21 noise, key = scico.random.randn(Cx.shape, seed=0)
22 y = Cx + sigma_p * noise
23
24 f = functional.ZeroFunctional()
25 g0 = loss.SquaredL2Loss(y=y)
26 lambda_p = 1.0e-4 # ell_p2,1 norm regularization parameter
27 g1 = lambda_p * functional.L21Norm()
28 g = functional.SeparableFunctional((g0, g1))
29
30 D = linop.FiniteDifference(input_shape=x_gt.shape, append=0)
31 A = linop.VerticalStack((C, D))
32
33 rho_p = 5.0e-2 # ADMM penalty parameter
34 maxiter = 500 # number of ADMM iterations
35 mu, nu = ProximalADMM.estimate_parameters(A)
36
37 solver = ProximalADMM(
38     f=f,
39     g=g,
40     A=A,
41     B=None,
42     rho=rho_p,
43     mu=mu,
44     nu=nu,
45     x0=C.adj(y),
46     maxiter=maxiter,
47     itstat_options={"display": True, "period": 10},

```

```

48 )
49
50 print(f"Solving on {device_info()}\n")
51 x = solver.solve()
52 hist = solver.itstat_object.history(transpose=True)
53
54 fig, ax = plot.subplots(nrows=1, ncols=3, figsize=(15, 5))
55 plot.imshow(x_gt, title="Ground truth", fig=fig, ax=ax[0])
56 nc = n // 2
57 yc = y[nc:-nc, nc:-nc]
58 plot.imshow(
59     solver.x, title="Deconvolved image: %.2f (dB)" % metric.psnr(x_gt,
60                     solver.x), fig=fig, ax=ax[2]
60 )
61 plot.imshow(y, fig=fig, ax=ax[1], title="Blurred image")
62 fig.show()

```

La implementación mostrada es una adaptación del código presentado en [Image Deconvolution with TV Regularization \(Proximal ADMM Solver\)](#)

Código A.8: Eliminación de ruido de sal y pimienta de una imagen con el modelo ROF

```

1 # Importamos las librerías necesarias
2
3 import scico.numpy as snp
4 from scico import functional, linop, loss, metric, plot
5 from scico.examples import spnoise
6 from scico.optimize.admm import ADMM, LinearSubproblemSolver
7 from scico.util import device_info
8 from scipy.ndimage import median_filter
9 plot.config_notebook_plotting()
10 from skimage import data, color
11
12 # Importamos la imagen de prueba y le añadimos ruido
13 imagen = color.rgb2gray(data.astronaut())
14 x_gt = imagen
15
16 x_gt = 0.5 * x_gt / x_gt.max()

```

```

17 y = spnoise(x_gt, 0.25)
18
19 # Definimos la función de costo y la regularización
20 lambda_p = 0.6e0
21 g_loss = loss.Loss(y=y, f=functional.L1Norm())
22 g_tv = lambda_p * functional.L21Norm()
23 # The append=0 option makes the results of horizontal and vertical
24 # finite
25 # differences the same shape, which is required for the L21Norm.
26 C = linop.FiniteDifference(input_shape=x_gt.shape, append=0)
27
28 solver = ADMM(
29     f=None,
30     g_list=[g_loss, g_tv],
31     C_list=[linop.Identity(input_shape=y.shape), C],
32     rho_list=[5e0, 5e0],
33     x0=y,
34     maxiter=500,
35     subproblem_solver=LinearSubproblemSolver(cg_kwargs={"tol": 1e-3, "maxiter": 20}),
36     itstat_options={"display": True, "period": 10},
37 )
38 print(f"Solving on {device_info()}\n")
39 x_tv = solver.solve()
40 hist = solver.itstat_object.history(transpose=True)
41
42 # Visualizamos los resultados
43 plt_args = dict(norm=plt.matplotlib.colors.Normalize(vmin=0, vmax=1.0))
44 fig, ax = plt.subplots(nrows=2, ncols=2, sharex=True, sharey=True,
45 figsize=(13, 12))
46 plt.imshow(x_gt, title="Ground truth", fig=fig, ax=ax[0, 0], **plt_args)
47 plt.imshow(y, title="Noisy image", fig=fig, ax=ax[0, 1], **plt_args)
48 plt.imshow(

```

```

49     title=f"Median filtering: {metric.psnr(x_gt, x_med):.2f} (dB)",
50     fig=fig,
51     ax=ax[1, 0],
52     **plt_args,
53 )
54 plot.imshow(
55     x_tv,
56     title=f"ell_1-TV denoising: {metric.psnr(x_gt, x_tv):.2f} (dB)",
57     fig=fig,
58     ax=ax[1, 1],
59     **plt_args,
60 )
61 fig.show()

```

La implementación mostrada es una adaptación del código presentado en [ℓ₁ Total Variation Denoising](#)

Código A.9: Generación de sinogramas con utilizando el módulo `scikit-image`

```

1 # Modulos
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from skimage.transform import radon
6 from skimage.data import shepp_logan_phantom
7
8 # Abrimos la imagen Circulo.png (Reemplazar por cualquier archivo de
9 # imagen)
10 imagen = plt.imread('Circulo.png')
11 # la convertimos a escala de grises
12 imagen = imagen[:, :, 0]
13 # invertimos la imagen
14 imagen = 1 - imagen
15
16 # Calculamos el sinograma
17 theta = np.linspace(0., 180., max(imagen.shape), endpoint=False)
18 sinogram = radon(imagen, theta=theta)

```

```
19 # Visualizar la imagen original y el sinograma
20 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
21
22 ax1.imshow(imagen, cmap=plt.cm.Greys_r)
23 ax2.imshow(sinogram, cmap=plt.cm.Greys_r, extent=(0, 180, 0, sinogram.
    shape[0]), aspect='auto')
24 ax1.axis('off')
25 ax2.axis('off')
26 plt.tight_layout()
27 plt.show()
28
29 # Guardamos la imagen generada
30 fig.savefig('sinograma_circulo.png', dpi=400)
```

A.2.1. Experimentación con CIL

Código A.10: Generación de geometrías paralelas 2D y 3D

```
1 # Importar librerías
2 from cil.framework import AcquisitionGeometry
3 from cil.utilities.display import show_geometry
4
5
6 # Definimos una geometría 2D con detector en el origen, de 10 píxeles y
7 # 180 ángulos.
8 ag_2d = AcquisitionGeometry.create_Parallel2D(detector_position=[0,0])
9 ag_2d.set_panel(num_pixels=10)
10 ag_2d.set_angles(angles=range(0,180))
11
12 # Mostramos la figura que ilustra la geometría
13 show_geometry(ag_2d)
14
15 # Mostramos la información detallada de la geometría
16 print(ag_2d)
17
18 # Definimos una geometría 3D con detector en el origen, de 10x15 píxeles
19 # y 180 ángulos.
20 ag_3d = AcquisitionGeometry.create_Parallel3D(detector_position=[0,0,0])
21 ag_3d.set_panel(num_pixels=[10,15])
22 ag_3d.set_angles(angles=range(0,180))
23
24 # Mostramos la figura que ilustra la geometría
25 show_geometry(ag_3d)
26
27
28
29 # Generamos una geometría de abanico 2D con fuente en (0,-5) y detector
# en (0,5)
```

```

30 ag_fan_beam = AcquisitionGeometry.create_Cone2D(source_position=[0,-5] ,
31     detector_position=[0,5])
32 ag_fan_beam.set_panel(num_pixels=5)
33 ag_fan_beam.set_angles(angles=range(0,180))
34
35 # Mostramos la figura que ilustra la geometría
36 show_geometry(ag_fan_beam)
37
38 # Mostramos la información detallada de la geometría
39 print(ag_fan_beam)
40
41 # Generamos una geometría de cono 3D con fuente en (0,-10,0) y detector
42     en (0,10,0)
43 ag_cone_beam = AcquisitionGeometry.create_Cone3D(source_position
44     =[0,-10,0],detector_position=[0,10,0])
45 ag_cone_beam.set_panel(num_pixels=[10,10])
46 ag_cone_beam.set_angles(angles=range(0,180))
47
48 # Mostramos la figura que ilustra la geometría
49 show_geometry(ag_cone_beam)
50
51 # Mostramos la información detallada de la geometría
52 print(ag_cone_beam)

```

Código A.11: Reconstrucción de datos sin procesar

```

1
2 # Definimos la ruta de los datos
3 path = '/home/daniellara/usb'
4
5 # Importamos las librerías
6 import os
7
8 from cil.io import ZEISSDataReader, TIFFWriter
9 from cil.processors import TransmissionAbsorptionConverter, Slicer
10 from cil.recon import FDK
11 from cil.utilities.display import show2D, show_geometry

```

```

12 from cil.utilities.jupyter import islicer, link_islicer
13
14 # Cargamos los datos
15 filename = os.path.join(path, "4_2014-03-20_1704_12/tomo-A/gruppe 4_tomo
   -A.txrm")
16
17 data = ZEISSDataReader(file_name=filename).read()
18
19 # Mostramos los datos de la geometría
20 print(data.geometry)
21
22 # Mostramos la gráfica de la geometría usada
23 show_geometry(data.geometry)
24
25 # Mostramos una de las proyecciones
26 show2D(data, slice_list=('angle', 210))
27
28 # Una versión interactiva de la proyección se obtiene con la siguiente
   linea
29 islicer(data)
30
31 # Utilizando TransmissionAbsorptionConverter para convertir de transmisi
   ón a absorción
32 data = TransmissionAbsorptionConverter()(data)
33
34 # Definimos el motor de reconstrucción, en este caso en particular
   utilizaremos "Tigre"
35 data.reorder(order='tigre')
36
37 # Definimos la geometría que se va a utilizar para la reconstrucción
38 ig = data.geometry.get_ImageGeometry()
39
40 # Realizamos la reconstrucción empleando el método FDK (Filtered back-
   projection)
41 fdk = FDK(data, ig)
42 recon = fdk.run()

```

```

43
44 # Mostramos un corte de la reconstrucción
45 show2D(recon, slice_list=[('vertical',875)])
46
47 # Una versión interactiva de la reconstrucción se obtiene con la
    siguiente linea
48 islicer(recon)

```

Código A.12: Preprocesamiento de datos con CIL

```

1 # Importamos las librerías
2 # cil imports
3 from cil.framework import ImageData, ImageGeometry
4 from cil.framework import AcquisitionGeometry, AcquisitionData
5
6 from cil.processors import (
7     CentreOfRotationCorrector,
8     Slicer,
9     Binner,
10    Masker,
11    MaskGenerator,
12    TransmissionAbsorptionConverter,
13 )
14
15 from cil.plugins.astra import FBP
16 from cil.utilities import dataexample
17 from cil.utilities.display import show2D, show_geometry
18
19 import numpy as np
20 import matplotlib.pyplot as plt
21 import logging
22
23 # Definimos el nivel de registros de CIL
24 logging.basicConfig(level=logging.WARNING)
25 cil_log_level = logging.getLogger('cil.processors')
26 cil_log_level.setLevel(logging.INFO)
27

```

```

28 # Definimos la configuración de colores para las gráficas
29 cmap = "gray"
30
31 # Cargamos los datos de ejemplo
32 data_raw = dataexample.SYNCHROTRON_PARALLEL_BEAM_DATA.get()
33
34 # Mostramos información detallada de la geometría
35 print(data_raw.geometry)
36
37 # Mostramos la gráfica de la geometría usada
38 show_geometry(data_raw.geometry)
39
40 # Visualizamos los datos sin procesar
41 show2D(data_raw, slice_list=(‘angle’,0), cmap=cmap, origin=‘upper-left’)
42
43 #####
44 # Conversión de transición a absorción
45 #####
46 # Reescalamos los datos a partir del promedio de una porción de los
    datos
47 background = data_raw.get_slice(vertical=20).mean()
48 data_raw /= background
49
50 # Convertimos los datos de transmisión a absorción
51 data_exp = TransmissionAbsorptionConverter()(data_raw)
52
53 # Graficamos los datos procesados
54 show2D(data_exp, slice_list=[(‘angle’,0), (‘angle’, 15), (‘angle’,30)],
      \
      cmap=cmap, num_cols=2, size=(15,15), origin=‘upper-left’)
55 #####
56 # Eliminación de pixeles brillantes
57 #####
58 # data_crop = Slicer(roi={‘vertical’: (1, None)})(data_exp)

```

```

62
63 show2D(data_crop, slice_list=[('angle',0), ('angle', 15), ('angle',30)],
64     \
64     cmap=cmap, num_cols=2, size=(15,15), origin='upper-left')
65
66 # Otro método más elaborado consiste en el manejo de estas situaciones
67 # como determinación de valores atípicos. El siguiente código muestra c
68 # ómo se puede realizar esto.
69
70
71 # Definimos el valor de umbral
72 mask = MaskGenerator.threshold(max_val=10)(data_exp)
73
74 # Aplicamos la máscara creada que contiene valores de 0 y 1 donde los
75 # valores atípicos fueron detectados y donde no, respectivamente.
76 data_masked = Masker.interpolate(mask=mask, method='nearest', axis='
77     vertical')(data_exp)
78
79 # Mostramos los resultados
80 show2D(data_masked, slice_list=[('angle',0), ('angle', 15), ('angle',30)
81     ],
82     \
82     cmap=cmap, num_cols=2, size=(15,15), origin='upper-left')
83
84 ######
85 # Corrección avanzada de píxeles defectuosos
86 #####
87
88 # Generamos artefactos en los datos
89
90 # Número de columnas con artefactos
91 number_of_columns = 3
92
93 # Número de píxeles sobreexpuestos ubicados aleatoriamente
94 number_of_hot_pix = 500
95
96 # Semilla para la generación de números
97 seed = 5932
98
99
100 data_corrupted = add_bad_pixels(data_slice, number_of_columns,

```

```

        number_of_hot_pix, seed)

92
93 show2D([data_slice, data_corrupted], \
94         ['clean data', 'corrupted data'], \
95         cmap=cmap, num_cols=2, size=(15,10))
96
97 # Reconstruimos los datos corruptos
98 fbp = FBP(ig_slice, data_slice.geometry, device='gpu')
99 fbp.set_input(data_slice)
100 fbp_recon_clean = fbp.get_output()
101
102 fbp.set_input(data_corrupted)
103 fbp_recon_corrupted = fbp.get_output()
104
105 show2D([fbp_recon_clean, fbp_recon_corrupted], \
106         ['clean data', 'corrupted data'], \
107         cmap=cmap, num_cols=2, size=(15,10), origin='upper-left')
108
109 # Definimos la máscara para los datos corruptos
110 mask = MaskGenerator.median(threshold_factor=3, window=7)(data_corrupted
111     )
112
113 show2D(mask)
114
115 # Aplicación de la máscara obtenida a los datos
116 data_masked = Masker.interpolate(mask=mask, method='linear', axis='
117     horizontal')(data_corrupted)
118
119 # visualise corrected sinogram
120 show2D([data_slice, data_masked, (data_slice-data_masked).abs()], \
121         ['clean data', 'masked data', 'difference'], \
122         cmap=cmap, num_cols=2, size=(15,15), origin='upper-left')
123
124 # Reconstrucción de los datos corregidos
125 fbp.set_input(data_masked)
126 fbp_recon_masked = fbp.get_output()

```

```

125
126 show2D([fbp_recon_clean, fbp_recon_masked], \
127     ['clean data', 'masked data'], \
128     cmap=cmap, num_cols=2, origin='upper-left')

```

Código A.13: Función para agregar artefactos

```

1 def add_bad_pixels(data, number_of_columns, number_of_hot_pix, seed):
2
3     data_corrupted = data.copy()
4
5     # Obtenemos el rango de intensidades de los datos
6     low = np.amin(data.as_array())
7     high = np.amax(data.as_array())
8
9     # Definimos la semilla para la generación de números aleatorios
10    rng = np.random.RandomState(seed=seed)
11    # Seleccionamos columnas aleatorias
12    columns = rng.randint(0, data.shape[1], size=number_of_columns)
13    # Seleccionamos píxeles aleatorios
14    pix_row = rng.randint(0, data.shape[0], size=number_of_hot_pix)
15    pix_col = rng.randint(0, data.shape[1], size=number_of_hot_pix)
16
17    pixel_values = rng.uniform(low=low, high=high, size=number_of_hot_pix
18        )
19
20    for i in range(number_of_columns):
21        col_pattern = rng.uniform(low=low, high=high, size=data.shape[0])
22        data_corrupted.as_array()[:, columns[i]] = data.as_array()[:, \
23            columns[i]]+col_pattern
24
25    for i in range(number_of_hot_pix):
26        data_corrupted.as_array()[pix_row[i], pix_col[i]] = pixel_values[
27            i]
28
29    return data_corrupted

```

Código A.14: Carga de datos para PDHG

```
1 # Importamos las librerías
2 from cil.framework import BlockDataContainer
3
4 from cil.optimisation.functions import L2NormSquared, L1Norm,
   BlockFunction, MixedL21Norm, IndicatorBox, TotalVariation
5 from cil.optimisation.operators import GradientOperator, BlockOperator
6 from cil.optimisation.algorithms import PDHG, SIRT
7
8 from cil.plugins.astra.operators import ProjectionOperator
9 from cil.plugins.astra.processors import FBP
10
11 from cil.plugins.ccpi_regularisation.functions import FGP_TV
12
13 from cil.utilities.display import show2D, show1D, show_geometry
14 from cil.utilities.jupyter import islicer
15
16 from cil.io import ZEISSDataReader
17
18 from cil.processors import Binner, TransmissionAbsorptionConverter,
   Slicer
19 import matplotlib.pyplot as plt
20 import numpy as np
21 import os
22
23 # Ruta de los datos
24 path = '/home/daniellara/egg1'
25
26 # Cargamos los datos
27 reader = ZEISSDataReader()
28 filename = os.path.join(path, "2014-03-20_946_13","tomo-A","gruppe 1
   _tomo-A.txrm")
29 data3D = ZEISSDataReader(file_name=filename).read()
30
31 # Mostramos el detalle de la geometría
32 print(ag3D)
```

```

33
34 # Mostramos la geometría
35 show_geometry(ag3D)
36
37 # Mostramos proyecciones de los datos
38 show2D(data3D, slice_list = [('vertical',512), ('angle',512), ('horizontal',512)], cmap="inferno", num_cols=3)

```

Código A.15: Reconstrucción con FBP

```

1 # Este código es una continuación del anterior
2
3 # Creamos los datos de absorción
4 data2D = data3D.get_slice(vertical=512, force=True)
5 absorption_data = TransmissionAbsorptionConverter()(data2D)
6
7 # Definimos la geometría de la reconstrucción
8 ag2D = absorption_data.geometry
9 ig2D = ag2D.get_ImageGeometry()
10
11 # Reconstrucción con FBP
12 fbp_recon = FBP(ig2D, ag2D, device = 'gpu')(absorption_data)
13
14 # Mostramos la reconstrucción
15 show2D(fbp_recon,
16         cmap = 'inferno',
17         slice_list=[('vertical',128),('vertical',256),('vertical',512),('vertical',640)],num_cols=2)
18
19 #####
20 # Simulamos datos incompletos para las reconstrucciones posteriores
21 #####
22
23 # Seleccionamos cada 10 ángulos
24 sliced_data = Slicer(roi={'angle':(0,1600,10)})(data2D)
25
26 # Create absorption data

```

```

27 absorption_data = TransmissionAbsorptionConverter()(sliced_data)
28
29 #####
30 # Reconstrucción con PDHG y regularización L1Norm
31 #####
32
33 A = ProjectionOperator(ig2D, ag2D, device = "gpu")
34
35 K = A
36 F = 0.5 * L2NormSquared(b=absorption_data)
37 alpha = 0.01
38 G = alpha * L1Norm()
39
40 pdhg_l1 = PDHG(f = F, g = G, operator = K,
41                 max_iteration = 500,
42                 update_objective_interval = 100)
43 pdhg_l1.run(verbose=1)

```

Código A.16: Reconstrucción con PDHG y regularización L^1

```

1 # Este código es una continuación del anterior
2
3 A = ProjectionOperator(ig2D, ag2D, device = "gpu")
4
5 K = A
6 F = 0.5 * L2NormSquared(b=absorption_data)
7 alpha = 0.01
8 G = alpha * L1Norm()
9
10 pdhg_l1 = PDHG(f = F, g = G, operator = K,
11                  max_iteration = 500,
12                  update_objective_interval = 100)
13 pdhg_l1.run(verbose=1)
14
15 # Mostramos la reconstrucción
16 show2D([pdhg_l1.solution, fbp_recon], fix_range=(0,0.05), title = ['L1
regularisation', 'FBP'], cmap = 'inferno')

```

Código A.17: Reconstrucción con PDHG y regularización de variación total

```
1 # Este código es una continuación del anterior
2
3 # Función por bloques para F
4 alpha_tv = 0.0003
5 f1 = alpha_tv * MixedL21Norm()
6 f2 = 0.5 * L2NormSquared(b=absorption_data)
7 F = BlockFunction(f1, f2)
8
9 # Función por bloques para K
10 Grad = GradientOperator(ig2D)
11 K = BlockOperator(Grad, A)
12
13 # Función G
14 G = IndicatorBox(lower=0)
15
16 pdhg_tv_explicit = PDHG(f = F, g = G, operator = K,
17                         max_iteration = 1000,
18                         update_objective_interval = 200)
19 pdhg_tv_explicit.run(verbose=1)
20
21 # Mostramos los resultados
22 show2D([pdhg_tv_explicit.solution, fbp_recon], fix_range=(0,0.055), title
= ['TV regularisation', 'FBP'], cmap = 'inferno')
```

Código A.18: Reconstrucción con PDHG y regularización de Tikhonov

```
1 # Define BlockFunction F
2 alpha_tikhonov = 0.05
3 f1 = alpha_tikhonov * L2NormSquared()
4 F = BlockFunction(f1, f2)
5
6 # Setup and run PDHG
7 pdhg_tikhonov_explicit = PDHG(f = F, g = G, operator = K,
8                                 max_iteration = 1000,
9                                 update_objective_interval = 200)
10 pdhg_tikhonov_explicit.run(verbose=1)
```

A.3. Experimentación con NiftyPET

Código A.19: Carga de datos con NiftyPET

```
1 # Importamos las librerias
2 import os, logging
3
4 from matplotlib.pyplot import (
5     matshow,
6     colorbar,
7     xlabel,
8     ylabel,
9     plot,
10    legend,
11    grid,
12 )
13
14 from niftypet import nipet
15
16 logging.basicConfig(level=logging.INFO)
17 mMRpars = nipet.get_mmrparams()
18
19 # Definimos la ruta a los archivos
20 folderin = '/mnt/test1dev/TIC/00Datos/amyloidPET_FBP_TPO'
21
22 # Leemos los archivos
23 datain = nipet.classify_input(folderin, mMRpars)
24
25 # Definimos la ruta de salida
26 oph = os.path.join( datain['corepath'], 'output')
27
28 datain
```

Código A.20: Extracción y generación de mapas de atenuación

```
1 # Obtenemos los mapas de atenuación de hardware (cama y bobina de cabeza
y cuello)
```

```

2 muhdct = nipet.hdw_mumap(datain, [1,2,4], mMRpars, outpath=opth,
   use_stored=True)
3
4 # Obtenemos el mapa de atenuación humano basado en RM
5 muodct = nipet.obj_mumap(datain, mMRpars, outpath=opth, store=True)
6
7 # Mapa de atenuación para el corte axial iz
8 iz = 75
9
10 # Mapa de atenuación para el equipo
11 matshow(muhdct['im'][iz,:,:], cmap='bone')
12 colorbar()
13
14 # Mapa de atenuación para el paciente
15 matshow(muodct['im'][iz,:,:], cmap='bone')
16 colorbar()
17
18 # Mapa de atenuación conjunto
19 matshow(muhdct['im'][iz,:,:] + muodct['im'][iz,:,:], cmap='bone')
20 colorbar()
21
22 # Mapa de atenuación para el corte axial iz_2
23 iz_2 = 115
24 matshow(muhdct['im'][iz_2,:,:] + muodct['im'][iz_2,:,:], cmap='bone')
25 colorbar()
26
27 # Mapa de atenuación para el corte sagital iz_3
28 ix = 150
29
30 # plot image with a colour bar
31 matshow(muhdct['im'][:, :, ix] + muodct['im'][:, :, ix], cmap='bone')
32 colorbar()

```

Código A.21: Obtención de los sinogramas

```

1 # Generamos los datos de sinograma a partir de los datos de entrada
2 hst = nipet.mmrhist(datain, mMRpars)

```

```

3
4 # Seleccionamos un corte para mostrar los sinogramas
5 si = 75
6
7 # Sinograma
8 matshow(hst['psino'][si,:,:], cmap='inferno')
9 colorbar()
10 xlabel('bins')
11 ylabel('angles')
12
13 # Sinograma retardado
14 matshow(hst['dsino'][si,:,:], cmap='inferno')
15 colorbar()
16 xlabel('bins')
17 ylabel('angles')
18
19 # Graficamos el recuento de eventos en función del tiempo
20 plot(hst['phc'], label='prompt')
21 plot(hst['dhc'], label='delayed')
22 legend()
23 grid('on')
24 xlabel('time')
25 ylabel('counts')

```

A.4. Tablas

Tabla A.1: Resultados de la experimentación para ML-EM

| Iteración | PSNR | SSIM | Tiempo (s) | Acumulado (s) |
|-----------|----------|---------|------------|---------------|
| 10 | 62,60636 | 0,99947 | 5,19696 | 5,19696 |
| 20 | 77,41000 | 0,99997 | 5,19583 | 10,39279 |
| 30 | 86,10061 | 0,99999 | 5,24160 | 15,63439 |
| 40 | 91,04850 | 1,00000 | 5,22539 | 20,85978 |
| 50 | 90,44401 | 0,99999 | 5,24898 | 26,10876 |

| Iteración | PSNR | SSIM | Tiempo (s) | Acumulado (s) |
|------------------|-------------|-------------|-------------------|----------------------|
| 60 | 93,82645 | 1,00000 | 5,25846 | 31,36722 |
| 70 | 96,21353 | 1,00000 | 5,28616 | 36,65338 |
| 80 | 97,92031 | 1,00000 | 5,27891 | 41,93230 |
| 90 | 99,24101 | 1,00000 | 5,25745 | 47,18975 |
| 100 | 100,35452 | 1,00000 | 5,26300 | 52,45276 |
| 110 | 101,33749 | 1,00000 | 5,29151 | 57,74427 |
| 120 | 102,20375 | 1,00000 | 5,25590 | 63,00017 |
| 130 | 102,93426 | 1,00000 | 5,26718 | 68,26735 |

Referencias bibliográficas

- [1] Bailey, D. L., Townsend, D. W., Valk, P. E., y Maisey, M. N. *Positron Emission Tomography: Basic Sciences*. Springer, 2005.
- [2] Ballinger, J. R. *PET Radiopharmaceuticals: Chemical, Biological, and Clinical Data*. Springer, 2022.
- [3] Barrett, H. H., Abbey, C. K., Karp, J. S., y Glick, M. R. List-mode likelihood. *Journal of the Optical Society of America*, 1997.
- [4] Barrett, H. H. y Myers, K. J. *Foundations of Imaging Science*. Wiley, 2004.
- [5] Barrett, H. H. y Swindell, W. *Radiological Imaging: The Theory of Image Formation, Detection, and Processing*. Academic Press, 1981.
- [6] Bertero, M., Boccacci, P., y Mol, C. D. *Introduction to Inverse Problems in Imaging*. CRC Press, 2022.
- [7] Brezis, H. Functional analysis, sobolev spaces and partial differential equations. *Springer*, 2011.
- [8] Brown, R., Kolbitsch, C., Delplancke, C., Papoutsellis, E., Mayer, J., Ovtchinnikov, E., Pasca, E., Neji, R., da Costa-Luis, C., Gillman, A. G., Ehrhardt, M. J., McClelland, J. R., Eiben, B., y Thielemans, K. Motion estimation and correction for simultaneous pet/mr using sirf and cil. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2204):20200208, July 2021.
- [9] Chambolle, A. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20:89–97, 2004.

- [10] Chambolle, A. y Pock, T. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, December 2010.
- [11] Chambolle, A. y Pock, T. An introduction to continuous optimization for imaging. *Acta Numerica*, 25:161–319, 2016.
- [12] Dempster, A. P., Laird, N. M., y Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.
- [13] Esser, E., Zhang, X., y Chan, T. F. A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. *SIAM Journal on Imaging Sciences*, 3(4):1015–1046, 2010.
- [14] Freeman, T. G. *The Mathematics of Medical Imaging*. Springer, 2015.
- [15] Folland, G. B. *Real Analysis: Modern Techniques and Their Applications*. Wiley, 1999.
- [16] Fraioli, F, editor. *PET/CT in Brain Disorders*. Springer International Publishing, 2019.
- [17] Galindo, P. X. Reconstrucción conjunta de imágenes de resonancia magnética y de tomografía por emisión de positrones utilizando datos sintéticos, 2023.
- [18] Grafakos, L. *Classical Fourier Analysis*. Springer, 2008.
- [19] Hadamard, J. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin*, 13:49–52, 1902.
- [20] Hudson, H. M. y Larkin, R. S. Accelerated image reconstruction using ordered subsets of projection data. *IEEE Transactions on Medical Imaging*, 13:601–609, 1994.
- [21] Jørgensen, J. S., Ametova, E., Burca, G., Fardell, G., Papoutsellis, E., Pasca, E., Thielemans, K., Turner, M., Warr, R., Lionheart, W. R. B., y Withers, P. J. Core imaging library - part i: a versatile python framework for tomographic imaging. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2204):20200192, July 2021.
- [22] Jørgensen, J. S., Andersen, M. S., y Gundlach, C. Hdtomo txrm micro-ct datasets, May 2021.

- [23] Kaipio, J. y Somersalo, E. *Statistical and Computational Inverse Problems*. Springer, New York, 1st edición, 2005.
- [24] Kak, A. C. y Slaney, M. *Principles of Computerized Tomographic Imaging*. Society for Industrial and Applied Mathematics, Philadelphia, 2001.
- [25] Lange, K. y Fessler, J. A. Em reconstruction algorithms for emission and transmission tomography. *Journal of Computer Assisted Tomography*, 1984.
- [26] Maier, A., Steidl, S., Christlein, V., y Hornegger, J. *Medical Imaging Systems: An Introductory Guide*. Springer Open, 2011.
- [27] Markiewicz, P. J., Cash, D., y Schott, J. M. Single amyloid PET scan on the Siemens Biograph mMR, November 2018.
- [28] Markiewicz, P. J., Cash, D., y Schott, J. M. Single amyloid PET scan on the Siemens Biograph mMR, June 2020.
- [29] Markiewicz, P. J., Ehrhardt, M. J., Erlandsson, K., Noonan, P. J., Barnes, A., Schott, J. M., Atkinson, D., Arridge, S. R., Hutton, B. F., y Ourselin, S. Niftynet: a high-throughput software platform for high quantitative accuracy and precision pet imaging and analysis. *Neuroinformatics*, 16(1):95–115, December 2017.
- [30] Mitrea, D. *Distributions, Partial Differential Equations, and Harmonic Analysis*. Springer, 2013.
- [31] Mueller, J. L. y Siltanen, S. *Linear and Nonlinear Inverse Problems with Practical Applications*. Society for Industrial and Applied Mathematics, Philadelphia, 1st edición, 2012.
- [32] Natterer, F. *The Mathematics of Computerized Tomography*. Society for Industrial and Applied Mathematics, New York, 1986.
- [33] Natterer, F. *Mathematical Methods in Image Reconstruction*. Society for Industrial and Applied Mathematics, Philadelphia, 2001.
- [34] Quarteroni, A., Sacco, R., y Saleri, F. *Numerical Mathematics*. Springer, 2007.
- [35] Richard, M. L. y Qi, J. Statistical approaches in quantitative positron emission tomography. *Statistics and Computing*, 2000.

- [36] Rudin, L., Osher, S., y Fatemi, E. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.
- [37] Shepp, L. A. y Vardi, Y. Maximum likelihood reconstruction for emission tomography. *IEEE Transactions on Medical Imaging*, 1:113–122, 1982.
- [38] Shiryaev, A. N. *Probability I*. Springer, 2016.
- [39] Wang, Z., Bovik, A. C., Sheikh, H. R., y Simoncelli, E. P. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4), 2004.
- [40] Zhu, M. y Chan, T. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. *CAM Report*, págs. 08–34, 2008.