Imágenes Médicas - Práctica 4: Reconstrucción de Imágenes Tomográficas: Métodos Iterativos

Juan Pablo Morales[1a]

[a] *Instituto Balseiro - Centro Atómico Bariloche. Av Bustillo 9500, 8400 Bariloche, Río Negro, Argentina*

25 de marzo de 2024

1. Introducción

En el proceso de reconstrucción de imágenes tomográficas, se busca reconstruir una imagen del objeto a partir de proyecciones. Entre los métodos de reconstrucción, los métodos iterativos son utilizados debido a su capacidad para producir imágenes de alta calidad y reducir artefactos de reconstrucción.

Los métodos iterativos como su nombre indica se basan en iterativamente para mejorar una estimación inicial de la imagen mediante la minimización de una función que compara las proyecciones medidas con las proyecciones reconstruidas a partir de la estimación actual de la imagen. Este proceso se repite hasta que se alcanza una convergencia satisfactoria.

2. El método ART

El Método de Reconstrucción Algebraica (ART) es una técnica ampliamente reconocida en el campo de la tomografía computarizada. Este método se utiliza para reconstruir imágenes a partir de proyecciones medidas. La idea principal detrás del ART es iterar sobre una estimación inicial de la imagen para mejorarla gradualmente hasta obtener una reconstrucción aceptable.

En el proceso de reconstrucción mediante ART, se realizan las siguientes etapas:

1. **Estimación inicial**: Se comienza con una estimación inicial de la imagen que se desea reconstruir.
2. **Medición de proyecciones**: Se miden las proyecciones de la imagen desde múltiples ángulos.
3. **Iteración**: Se lleva a cabo un proceso iterativo para mejorar la estimación de la imagen. La ecuación fundamental que define este proceso es:

$$f_j^{(k+1)} = f_j^{(k)} + \frac{p_i - \sum_{j=1}^{N} f_{ji}^{(k)}}{N} \tag{1}$$

Donde:

- $f_j^{(k)}$ es la estimación actual de la intensidad de píxel en la posición j en la iteración k.
- $f_j^{(k+1)}$ es la estimación propuesta para la intensidad de píxel en la siguiente iteración.
- p_i es la proyección medida a lo largo de la dirección i.
- N es el número total de píxeles en la dirección considerada.
- $\sum_{j=1}^{N} f_{ji}^{(k)}$ es la suma de las intensidades de píxel a lo largo de la dirección i en la iteración k.

Teniendo en cuenta el algoritmo ART, se realizó una reconstrucción de una imagen de tamaño 2x2 con un total de 6 mediciones p_i. Esta reconstrucción se llevó a cabo en 3 pasos, como se muestra en la Figura 1. Inicialmente, se tomaron las mediciones en la dirección horizontal (p_1 y p_2). Partiendo de la suposición inicial de que $f_j^{(0)} = 0 \;\forall\; j$, se aplicó la ecuación (1) para realizar el primer paso de la reconstrucción.

Posteriormente, se llevaron a cabo mediciones en la dirección vertical (p_4 y p_5), continuando desde el paso anterior. Finalmente, se realizaron mediciones en la dirección diagonal (p_3 y p_6). Se observó que la reconstrucción convergió y la imagen resultante satisface las mediciones en todas las direcciones.

[1] juan.morales@ib.edu.ar

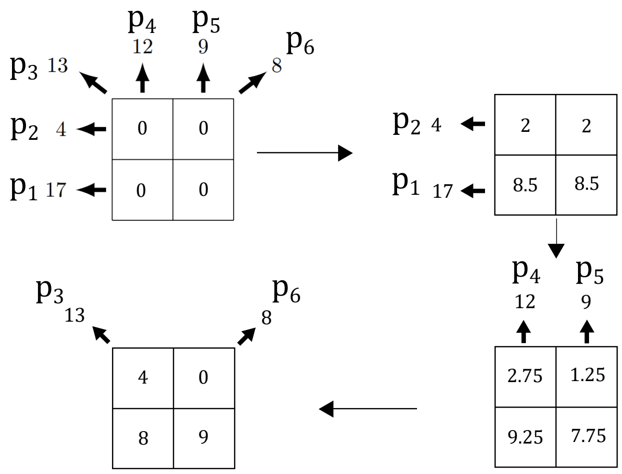

Figura 1: Pasos de reconstrucción por el método ART de una imagen de 2x2 con 6 mediciones en las direcciones horizontal, diagonal y vertical.

3. Dependencias de las reconstrucciones

Se llevó a cabo un análisis de la influencia del número de ángulos, detectores y nivel de ruido en las reconstrucciones realizadas mediante el método de reconstrucción iterativa ART utilizando MATLAB y los archivos de AIRTools. En primer lugar, se generó un fantoma Shepp-Logan (ver Figura 2) con un tamaño de $N \times N$ píxeles, donde $N = 256$, junto con su correspondiente sinograma. Posteriormente, se realizaron las reconstrucciones utilizando MATLAB, y mediante Python, se calcularon los errores de reconstrucción. En este caso, se utilizaron las siguientes métricas:

- Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$$

- Peak Signal to Noise Ratio (PSNR):

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX^2}{MSE}\right)$$

  , donde MAX es el máximo valor posible de la imagen. En este caso 255.

- Structural Similarity Index (SSIM):

$$SSIM = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

  , donde μ_x es la media muestral de píxeles de x; μ_y es la media muestral de píxeles de y; σ_x^2 es la varianza de x; σ_y^2 es la varianza de y; σ_{xy} es la covarianza de x e y; $c_1 = (k_1L)^2$, $c_2 = (k_2L)^2$ son dos variables para estabilizar la división con denominador débil; L es el rango dinámico de los valores de píxeles (típicamente esto es $2^{\#bits\ per\ pixel} - 1$); $k_1 = 0{,}01$ y $k_2 = 0{,}03$ por defecto.

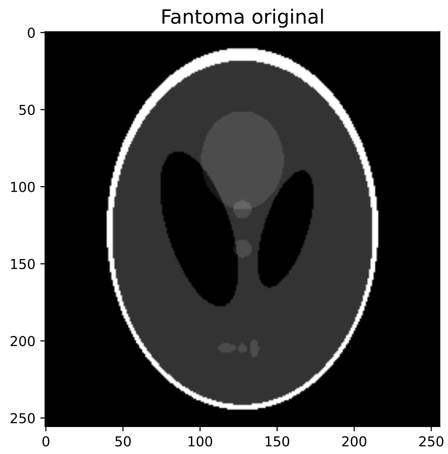

Figura 2: Fantoma original shepp-logan utilizado.

En primer lugar, se fijaron el número de detectores en $p = 360$, el número de iteraciones en $k = 100$, y el error relativo en $\eta = 1 \times 10^{-4}$. Este error relativo se define como el error en las mediciones comparado con $||p_i||$. Luego, se varió la cantidad de ángulos, tomando valores $\theta \in [0, 180]$ y variando el número de ángulos en 36, 60, 180 y 360. Los resultados obtenidos para las distintas reconstrucciones se muestran en la Figura 3. En esta figura, se puede observar que a medida que se aumenta el número de ángulos medidos, se obtiene una mejor calidad en la reconstrucción de la imagen.

Para tener una medida correcta del error de reconstrucción, se utilizaron las métricas mencionadas anteriormente. Estas métricas se muestran en la Figura 4 en función de la cantidad de ángulos utilizados.

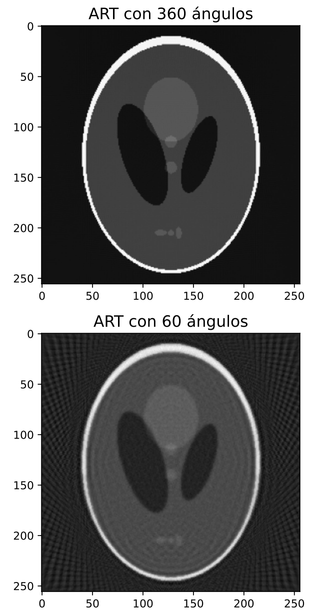

Figura 3: Reconstrucciones del fantoma shepp-logan mediante el método ART variando el número de ángulos utilizados.

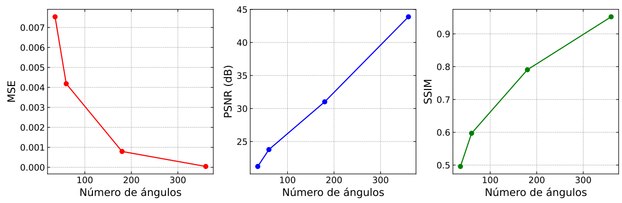

Figura 4: Medidas MSE, PSNR y SSIM del error de reconstrucción del fantoma shepp-logan mediante el método ART variando el número de ángulos utilizados.

Para el análisis del número de detectores, se varió el número tomando $p = \{60,\ 90,\ 180,\ 360\}$. Se mantuvieron los parámetros del caso anterior: el número de iteraciones $k = 100$ y el error relativo $\eta = 1 \times 10^{-4}$, y se fijó el número de ángulos $\theta \in [0,\ 180]$ en 360.

Los resultados se muestran en la Figura 5. En esta figura, se puede observar que a medida que se aumenta el número de detectores, se obtiene una mejor reconstrucción, y que su efecto es similar al que ocurre con el número de ángulos, incluyendo artefactos en la imagen reconstruida. Esto se evidencia también en las métricas mostradas en la Figura 6.

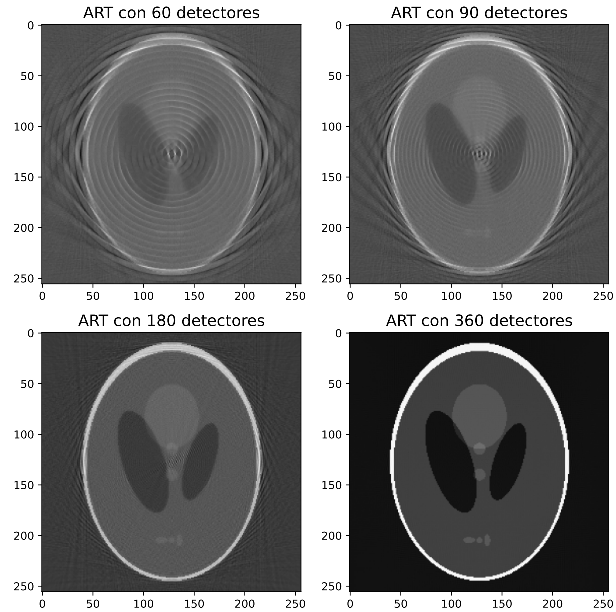

Figura 5: Reconstrucciones del fantoma shepp-logan mediante el método ART variando el número de detectores utilizados.

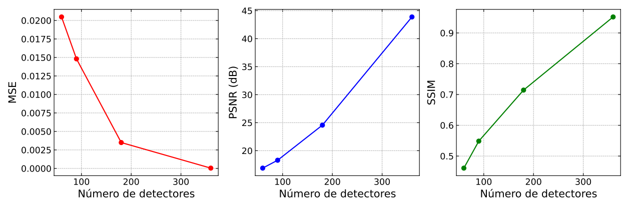

Figura 6: Medidas MSE, PSNE, SSIM del error de reconstrucción del fantoma shepp-logan mediante el método ART variando el número de detectores utilizados.

Además, se realizó el mismo estudio variando el nivel de intensidad del ruido. Se dejó el número de detectores en $p = 360$, el número de iteraciones en $k = 100$, y el número de ángulos $\theta \in [0,\ 180]$ en 360. Los valores utilizados para variar el ruido fueron los del ruido relativo $\eta = \{0{,}1,\ 0{,}01,\ 0{,}001,\ 0{,}0001,\ 0{,}00001\}$.

Los resultados se muestran en la Figura 7. Se puede observar que, si bien el ruido no añade artefactos a la reconstrucción, sí cambia la información que se puede obtener de las imágenes, ya que se pierden detalles al aumentar el ruido relativo. Al igual que en los casos anteriores, esto se evidencia en las métricas del error de reconstrucción mostradas en la Figura 7, donde a medida que aumenta el ruido, las métricas empeoran. Sin embargo, para valores como 0,0001 y 0,00001, la evaluación del error de reconstrucción es similar.

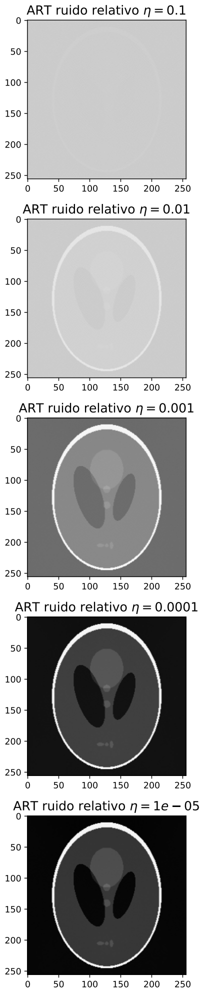

Figura 7: Reconstrucciones del fantoma shepp-logan mediante el método ART variando el nivel de ruido relativo.

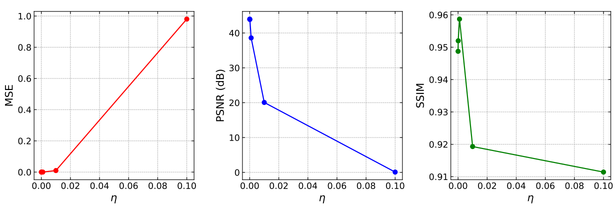

Figura 8: Medidas MSE, PSNE, SSIM del error de reconstrucción del fantoma shepp-logan mediante el método ART variando el nivel de ruido relativo η.

4. Análisis para distintos métodos

Se realizó el mismo análisis con respecto a las variables que en el caso anterior, pero esta vez para tres métodos alternativos de ART:

- **Simétrico**: Se realiza una barrida en un sentido seguida de una segunda en el sentido opuesto.
- **Aleatorio**: Se elige el índice i para las mediciones de manera aleatoria.
- **Simultáneo (SART)**: Se actualizan todos los índices simultáneamente usando la siguiente fórmula:

$$f^{(k+1)} = f^{(k)} + \mu T A^T M (p - A f^{(k)}), \quad (2)$$

donde $T = V^{-1}$ y $M = W^{-1}$, con V y W siendo matrices diagonales cuyos elementos son las sumas de filas y columnas de A.

Se utilizó el mismo fantoma Shepp-Logan de tamaño 256×256 y los mismos valores utilizados en la exploración de ART anteriormente.

En el análisis de variar la cantidad de ángulos utilizados, se obtuvieron los resultados mostrados en la Figura 9 para las métricas anteriormente analizadas. Se puede observar que, mientras que los métodos ART, ART simétrico y ART aleatorio dan resultados similares, el método simultáneo llega rápidamente a un valor en el cual no mejora la reconstrucción, y según las métricas, esta reconstrucción es peor que la de los otros métodos.

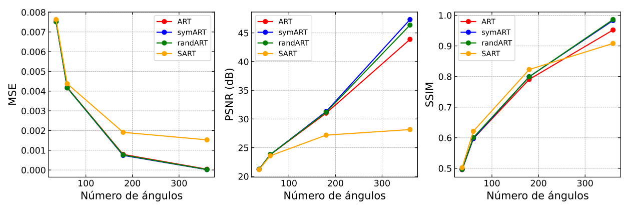

Figura 9: Medidas MSE, PSNE, SSIM del error de reconstrucción del fantoma shepp-logan mediante los métodos de reconstrucción ART, ART simétrico, ART aleatorio y SART variando el número de ángulos.

El análisis para el número de detectores se repite y se obtienen los resultados mostrados en la

Figura 10. En este caso, se observa un comportamiento similar al anterior, en el cual el método que obtiene los peores resultados es el método SART. Sin embargo, la diferencia entre este y los otros métodos es menor que en el caso de variar el número de ángulos, por lo tanto, afecta menos a esta reconstrucción.

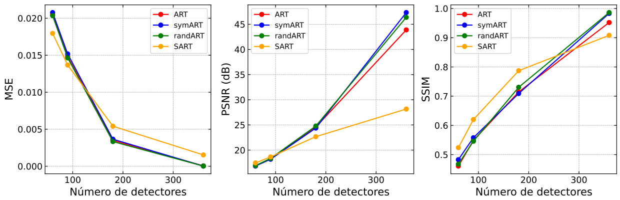

Figura 10: Medidas MSE, PSNE, SSIM del error de reconstrucción del fantoma shepp-logan mediante los métodos de reconstrucción ART, ART simétrico, ART aleatorio y SART variando el número de detectores.

El análisis del error de reconstrucción, variando el ruido relativo, se muestra en la Figura 11. Este caso resulta interesante debido a que, según los errores de reconstrucción dados por el MSE y el PSNR, el método SART muestra un mejor rendimiento, seguido por el método ART aleatorio, y luego por los métodos ART simétrico y ART, en contraposición a los análisis realizados anteriormente. Sin embargo, se observa que el método SART produce resultados peores que los otros métodos cuando el ruido relativo es aproximadamente menor a $\eta = 0{,}01$.

Además, es importante destacar que la métrica SSIM no refleja esta última observación, lo que sugiere que se pierden correlaciones con el método SSIM. Sin embargo, esto no implica necesariamente un rendimiento inferior en presencia de ruido (ver Figura 12).

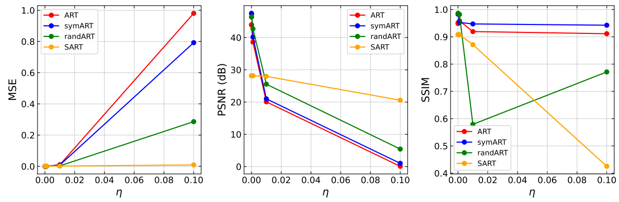

Figura 11: Medidas MSE, PSNE, SSIM del error de reconstrucción del fantoma shepp-logan mediante los métodos de reconstrucción ART, ART simétrico, ART aleatorio y SART variando el valor del ruido relativo.

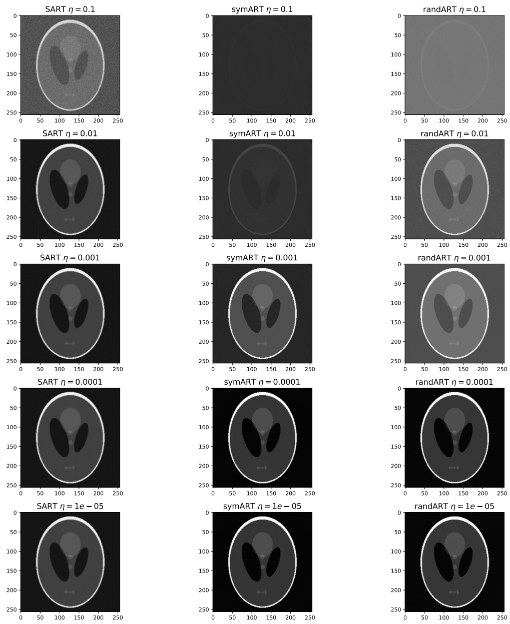

Figura 12: Reconstrucciones del fantoma shepp-logan mediante los métodos de reconstrucción ART, ART simétrico, ART aleatorio y SART variando el valor del ruido relativo.

5. Tiempos de cálculo

Para comparar los tiempos de cálculo, se realizó la medición del tiempo de ejecución de cada uno de estos métodos en MATLAB, y se compararon también con el método de retroproyección filtrada en Python. Estas mediciones se llevaron a cabo utilizando el mismo conjunto de parámetros: 360 detectores, 100 iteraciones para los métodos iterativos, sin ruido y 360 ángulos entre 0 y 180 grados. Los resultados obtenidos se muestran en la tabla 1.

Se observa que los métodos iterativos, en promedio, requieren mucho más tiempo en comparación con el método directo de la retroproyección filtrada. Esto se debe a que los métodos iterativos implican un mayor esfuerzo computacional, ya que requieren múltiples iteraciones para converger hacia la solución.

Método	Segundos
ART	590
ART Simetrico	1170
ART Aleatorio	598
SART	5.8
Retroproyección filtrada	0.2

Tabla 1: Tabla con los métodos de reconstrucción y el tiempo en segundos que llevó realizarlo.

6. Expectation Maximization

El método de Expectation Maximization (EM) se basa en la reconstrucción de la función $\vec{f}$ maximizando la probabilidad de obtener las mediciones $\vec{p}$ dado $\vec{f}$. Para lograr esto, se busca una cota inferior de esta probabilidad que pueda ser maximizada mediante iteraciones. De esta manera, se obtiene la reconstrucción de $\vec{f}$ deseada.

Se llevó a cabo un análisis de la dependencia del error de reconstrucción utilizando el algoritmo EM en relación con el número de iteraciones realizadas. Este error de reconstrucción se evaluó utilizando las mismas métricas utilizadas anteriormente. Algunos resultados de las reconstrucciones se muestran en la Figura 13. Se observa que, al aumentar las iteraciones, la calidad de la reconstrucción mejora; sin embargo, llega un punto en el que aumentar el número de iteraciones empeora la calidad de la reconstrucción. Este efecto se puede observar en la Figura 14, donde se muestran las métricas en función del número de iteraciones realizadas. Esto se observa claramente para las métricas MSE y PSNE, sin embargo, el error de reconstrucción calculado con la métrica SSIM no logra captar este suceso.

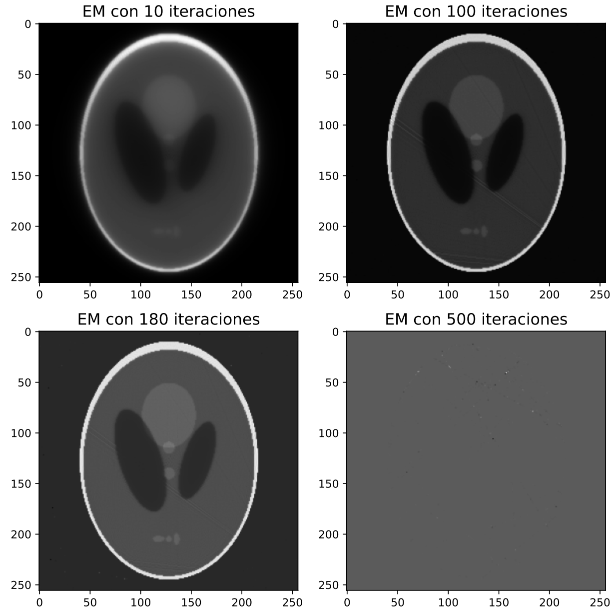

Figura 13: Reconstrucciones del fantoma shepp-logan mediante el método de reconstrucción expectation maximization variando el número de iteraciones realizado.

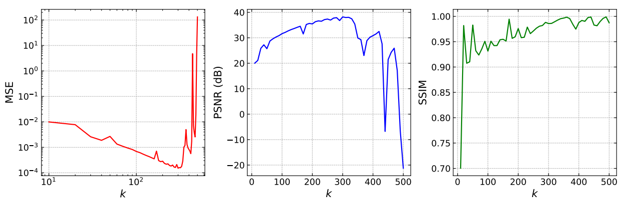

Figura 14: Medidas MSE, PSNE, SSIM del error de reconstrucción del fantoma shepp-logan mediante el métod de reconstrucción expectation maximization variando el número de repeticiones realizado.

7. Código

```
# # Ejercicio 2

# %%
import os
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import radon, iradon
from scipy.io import loadmat
from scipy.stats import poisson
from skimage.metrics import mean_squared_error as mse
from skimage.metrics import peak_signal_noise_ratio as psnr
from skimage.metrics import structural_similarity as ssim
import cv2

# %%
path = r"C:/Users/Propietario/Desktop/ib/5-Maestría/Imágenes Médicas/
    Practicas/Practica4/AIRtools/"

# %%
files_angulos = ["ART_angles0.5.mat", "ART_angles1.mat", "ART_angles3.
    mat", "ART_angles5.mat"]
num_angulos = [360, 180, 60, 36]
fantoma_path = os.path.join(path, "fantoma256.mat")
fantoma_original = loadmat(fantoma_path)['fantoma']
plt.imshow(fantoma_original, cmap='gray')
plt.title('Fantoma original', fontsize='x-large')
plt.tight_layout()
plt.savefig('fantoma_original.pdf')
plt.show()

# %%
fig, ax = plt.subplots(2,2,figsize=(8, 8))
for i, (n, file) in enumerate(zip(num_angulos, files_angulos)):
    path_actual = os.path.join(path, file)
    reconstruccion_fantoma = loadmat(path_actual)['ART']
    if i == 0 or i == 1:
        ax[0,i].imshow(reconstruccion_fantoma, cmap='gray')
        ax[0,i].set_title('ART con {} ángulos'.format(n), fontsize='x-
            large')
    else:
        ax[1,i%2].imshow(reconstruccion_fantoma, cmap='gray')
        ax[1,i%2].set_title('ART con {} ángulos'.format(n), fontsize='x
            -large')
plt.tight_layout()
plt.savefig('ART_angulos.pdf')
plt.show()

# %%
mse_a = []
psnr_a = []
```

```python
ssim_a = []
for n, file in zip(num_angulos, files_angulos):
    path_actual = os.path.join(path, file)
    reconstruccion_fantoma = loadmat(path_actual)['ART']
    mse_a.append(mse(fantoma_original, reconstruccion_fantoma))
    psnr_a.append(psnr(fantoma_original, reconstruccion_fantoma))
    ssim_a.append(ssim(fantoma_original, reconstruccion_fantoma, data_range=reconstruccion_fantoma.max()-reconstruccion_fantoma.min()))


# %%
files_detectores = ["ART_detectorsv260.mat", "ART_detectorsv290.mat", "ART_detectorsv2180.mat", "ART_detectorsv2360.mat"]
num_detectores = [60, 90, 180, 360]
fig, ax = plt.subplots(2,2,figsize=(8, 8))
for i, (n, file) in enumerate(zip(num_detectores, files_detectores)):
    path_actual = os.path.join(path, file)
    reconstruccion_fantoma = loadmat(path_actual)['ART']
    if i == 0 or i == 1:
        ax[0,i].imshow(reconstruccion_fantoma, cmap='gray')
        ax[0,i].set_title('ART con {} detectores'.format(n), fontsize='x-large')
    else:
        ax[1,i%2].imshow(reconstruccion_fantoma, cmap='gray')
        ax[1,i%2].set_title('ART con {} detectores'.format(n), fontsize='x-large')
plt.tight_layout()
plt.savefig('ART_detectores.pdf')
plt.show()

# %%
mse_detectors = []
psnr_detectors = []
ssim_detectors = []
for n, file in zip(num_detectores, files_detectores):
    path_actual = os.path.join(path, file)
    reconstruccion_fantoma = loadmat(path_actual)['ART']
    mse_detectors.append(mse(fantoma_original, reconstruccion_fantoma))
    psnr_detectors.append(psnr(fantoma_original, reconstruccion_fantoma))
    ssim_detectors.append(ssim(fantoma_original, reconstruccion_fantoma, data_range=reconstruccion_fantoma.max()-reconstruccion_fantoma.min()))


# %%
files_ruido = ["ART_rnoise0.1.mat", "ART_rnoise0.01.mat", "ART_rnoise0.001.mat", "ART_rnoise0.0001.mat", "ART_rnoise1e-05.mat"]
eta_list = [0.1, 0.01, 0.001, 1e-4, 1e-5]
fig, ax = plt.subplots(5,1,figsize=(10, 16))
for i, (eta, file) in enumerate(zip(eta_list, files_ruido)):
```

```
    path_actual = os.path.join(path, file)
    reconstruccion_fantoma = loadmat(path_actual)['ART']
    ax[i].imshow(reconstruccion_fantoma, cmap='gray')
    ax[i].set_title('ART ruido relativo $\eta = {}$'.format(eta),
       fontsize='x-large')
plt.tight_layout()
plt.savefig('ART_rnoise.pdf')
plt.show()

# %%
mse_noise = []
psnr_noise = []
ssim_noise = []
for n, file in zip(eta_list, files_ruido):
    path_actual = os.path.join(path, file)
    reconstruccion_fantoma = loadmat(path_actual)['ART']
    mse_noise.append(mse(fantoma_original, reconstruccion_fantoma))
    psnr_noise.append(psnr(fantoma_original, reconstruccion_fantoma))
    ssim_noise.append(ssim(fantoma_original, reconstruccion_fantoma,
       data_range=reconstruccion_fantoma.max()-reconstruccion_fantoma.
       min()))


# %%
files_angulos_sym = ["symART_angles0.5.mat", "symART_angles1.mat", "
   symART_angles3.mat", "symART_angles5.mat"]
files_angulos_rand = ["randART_angles0.5.mat", "randART_angles1.mat", "
   randART_angles3.mat", "randART_angles5.mat"]
files_angulos_sart = ["SART_angles0.5.mat", "SART_angles1.mat", "
   SART_angles3.mat", "SART_angles5.mat"]
# path_test = os.path.join(path, "symART_angles0.5.mat")
# print(loadmat(path_test).keys())

# %%
mse_a = []
mse_sym = []
mse_rand = []
mse_sart = []
psnr_a = []
psnr_sym = []
psnr_rand = []
psnr_sart = []
ssim_a = []
ssim_sym = []
ssim_rand = []
ssim_sart = []

for file_art, file_sym, file_rand, file_sart in zip( files_angulos,
   files_angulos_sym, files_angulos_rand, files_angulos_sart):
    path_actual = os.path.join(path, file_art)
    path_sym = os.path.join(path, file_sym)
    path_rand = os.path.join(path, file_rand)
```

```python
    path_sart = os.path.join(path, file_sart)

    reconstruccion_art = loadmat(path_actual)['ART']
    reconstruccion_sym = loadmat(path_sym)['symART']
    reconstruccion_rand = loadmat(path_rand)['randART']
    reconstruccion_sart = loadmat(path_sart)['SART']

    mse_a.append(mse(fantoma_original, reconstruccion_art))
    psnr_a.append(psnr(fantoma_original, reconstruccion_art))
    ssim_a.append(ssim(fantoma_original, reconstruccion_art, data_range=reconstruccion_art.max()-reconstruccion_art.min()))

    mse_sym.append(mse(fantoma_original, reconstruccion_sym))
    psnr_sym.append(psnr(fantoma_original, reconstruccion_sym))
    ssim_sym.append(ssim(fantoma_original, reconstruccion_sym, data_range=reconstruccion_sym.max()-reconstruccion_sym.min()))

    mse_rand.append(mse(fantoma_original, reconstruccion_rand))
    psnr_rand.append(psnr(fantoma_original, reconstruccion_rand))
    ssim_rand.append(ssim(fantoma_original, reconstruccion_rand, data_range=reconstruccion_rand.max()-reconstruccion_rand.min()))

    mse_sart.append(mse(fantoma_original, reconstruccion_sart))
    psnr_sart.append(psnr(fantoma_original, reconstruccion_sart))
    ssim_sart.append(ssim(fantoma_original, reconstruccion_sart, data_range=reconstruccion_sart.max()-reconstruccion_sart.min()))

# %% [markdown]
# ### Variando los detectores

# %%
files_detectores = ["ART_detectorsv260.mat", "ART_detectorsv290.mat", "ART_detectorsv2180.mat", "ART_detectorsv2360.mat"]
files_detectores_sym = ["symART_detectorsv260.mat", "symART_detectorsv290.mat", "symART_detectorsv2180.mat", "symART_detectorsv2360.mat"]
files_detectores_rand = ["randART_detectorsv260.mat", "randART_detectorsv290.mat", "randART_detectorsv2180.mat", "randART_detectorsv2360.mat"]
files_detectores_sart = ["SART_detectorsv260.mat", "SART_detectorsv290.mat", "SART_detectorsv2180.mat", "SART_detectorsv2360.mat"]
num_detectores = [60, 90, 180, 360]

# %%
mse_a = []
mse_sym = []
mse_rand = []
mse_sart = []
psnr_a = []
psnr_sym = []
psnr_rand = []
psnr_sart = []
```

```python
ssim_a = []
ssim_sym = []
ssim_rand = []
ssim_sart = []

for file_art, file_sym, file_rand, file_sart in zip( files_detectores, files_detectores_sym, files_detectores_rand, files_detectores_sart):
    path_actual = os.path.join(path, file_art)
    path_sym = os.path.join(path, file_sym)
    path_rand = os.path.join(path, file_rand)
    path_sart = os.path.join(path, file_sart)

    reconstruccion_art = loadmat(path_actual)['ART']
    reconstruccion_sym = loadmat(path_sym)['symART']
    reconstruccion_rand = loadmat(path_rand)['randART']
    reconstruccion_sart = loadmat(path_sart)['SART']

    mse_a.append(mse(fantoma_original, reconstruccion_art))
    psnr_a.append(psnr(fantoma_original, reconstruccion_art))
    ssim_a.append(ssim(fantoma_original, reconstruccion_art, data_range=reconstruccion_art.max()-reconstruccion_art.min()))

    mse_sym.append(mse(fantoma_original, reconstruccion_sym))
    psnr_sym.append(psnr(fantoma_original, reconstruccion_sym))
    ssim_sym.append(ssim(fantoma_original, reconstruccion_sym, data_range=reconstruccion_sym.max()-reconstruccion_sym.min()))

    mse_rand.append(mse(fantoma_original, reconstruccion_rand))
    psnr_rand.append(psnr(fantoma_original, reconstruccion_rand))
    ssim_rand.append(ssim(fantoma_original, reconstruccion_rand, data_range=reconstruccion_rand.max()-reconstruccion_rand.min()))

    mse_sart.append(mse(fantoma_original, reconstruccion_sart))
    psnr_sart.append(psnr(fantoma_original, reconstruccion_sart))
    ssim_sart.append(ssim(fantoma_original, reconstruccion_sart, data_range=reconstruccion_sart.max()-reconstruccion_sart.min()))


# %%
files_ruido = ["ART_rnoise0.1.mat", "ART_rnoise0.01.mat", "ART_rnoise0.001.mat", "ART_rnoise0.0001.mat", "ART_rnoise1e-05.mat"]
files_ruido_sym = ["symART_rnoise0.1.mat", "symART_rnoise0.01.mat", "symART_rnoise0.001.mat", "symART_rnoise0.0001.mat", "symART_rnoise1e-05.mat"]
files_ruido_rand = ["randART_rnoise0.1.mat", "randART_rnoise0.01.mat", "randART_rnoise0.001.mat", "randART_rnoise0.0001.mat", "randART_rnoise1e-05.mat"]
files_ruido_sart = ["SART_rnoise0.1.mat", "SART_rnoise0.01.mat", "SART_rnoise0.001.mat", "SART_rnoise0.0001.mat", "SART_rnoise1e-05.mat"]
eta_list = [0.1, 0.01, 0.001, 1e-4, 1e-5]

```

```python
# %%
mse_a = []
mse_sym = []
mse_rand = []
mse_sart = []
psnr_a = []
psnr_sym = []
psnr_rand = []
psnr_sart = []
ssim_a = []
ssim_sym = []
ssim_rand = []
ssim_sart = []

for file_art, file_sym, file_rand, file_sart in zip(files_ruido,
    files_ruido_sym, files_ruido_rand, files_ruido_sart):
    path_actual = os.path.join(path, file_art)
    path_sym = os.path.join(path, file_sym)
    path_rand = os.path.join(path, file_rand)
    path_sart = os.path.join(path, file_sart)

    reconstruccion_art = loadmat(path_actual)['ART']
    reconstruccion_sym = loadmat(path_sym)['symART']
    reconstruccion_rand = loadmat(path_rand)['randART']
    reconstruccion_sart = loadmat(path_sart)['SART']

    mse_a.append(mse(fantoma_original, reconstruccion_art))
    psnr_a.append(psnr(fantoma_original, reconstruccion_art))
    ssim_a.append(ssim(fantoma_original, reconstruccion_art, data_range
        =reconstruccion_art.max()-reconstruccion_art.min()))

    mse_sym.append(mse(fantoma_original, reconstruccion_sym))
    psnr_sym.append(psnr(fantoma_original, reconstruccion_sym))
    ssim_sym.append(ssim(fantoma_original, reconstruccion_sym,
        data_range=reconstruccion_sym.max()-reconstruccion_sym.min()))

    mse_rand.append(mse(fantoma_original, reconstruccion_rand))
    psnr_rand.append(psnr(fantoma_original, reconstruccion_rand))
    ssim_rand.append(ssim(fantoma_original, reconstruccion_rand,
        data_range=reconstruccion_rand.max()-reconstruccion_rand.min()))

    mse_sart.append(mse(fantoma_original, reconstruccion_sart))
    psnr_sart.append(psnr(fantoma_original, reconstruccion_sart))
    ssim_sart.append(ssim(fantoma_original, reconstruccion_sart,
        data_range=reconstruccion_sart.max()-reconstruccion_sart.min()))

# %%
eta_list = [0.1, 0.01, 0.001, 1e-4, 1e-5]
fig, ax = plt.subplots(5,3,figsize=(16, 16))
for i, (eta, sart, sym, rand) in enumerate(zip(eta_list,
    files_ruido_sart, files_ruido_sym, files_ruido_rand)):
    path_sart = os.path.join(path, sart)
```

```python
    path_sym = os.path.join(path, sym)
    path_rand = os.path.join(path, rand)
    reconstruccion_sart = loadmat(path_sart)['SART']
    reconstruccion_sym = loadmat(path_sym)['symART']
    reconstruccion_rand = loadmat(path_rand)['randART']
    ax[i,0].imshow(reconstruccion_sart, cmap='gray')
    ax[i,0].set_title('SART $\eta = {}$'.format(eta), fontsize='x-large
        ')
    ax[i,1].imshow(reconstruccion_sym, cmap='gray')
    ax[i,1].set_title('symART $\eta = {}$'.format(eta), fontsize='x-
        large')
    ax[i,2].imshow(reconstruccion_rand, cmap='gray')
    ax[i,2].set_title('randART $\eta = {}$'.format(eta), fontsize='x-
        large')
plt.tight_layout()
plt.savefig('rnoise.pdf')
plt.show()

# %% [markdown]
# ## Expectation maximization

# %%
files_em_show = ["EM_k10.mat", "EM_k100.mat", "EM_k180.mat", "EM_k500.
    mat"]

num_k = list(np.arange(10,510,10))
files_em = []
for k in num_k:
    files_em.append("EM_k{}.mat".format(k))
num_k_show = [10,100,180,500]

# %%
fig, ax = plt.subplots(2,2,figsize=(8, 8))
for i, (k, file) in enumerate(zip(num_k_show, files_em_show)):
    path_actual = os.path.join(path, file)
    reconstruccion_fantoma = loadmat(path_actual)['EMr']
    if i == 0 or i == 1:
        ax[0,i].imshow(reconstruccion_fantoma, cmap='gray')
        ax[0,i].set_title('EM con {} iteraciones'.format(k), fontsize='
            x-large')
    else:
        ax[1,i%2].imshow(reconstruccion_fantoma, cmap='gray')
        ax[1,i%2].set_title('EM con {} iteraciones'.format(k), fontsize
            ='x-large')
plt.tight_layout()
plt.savefig('EM_iteraciones.pdf')
plt.show()

# %%
mse_em = []
psnr_em = []
ssim_em = []
```

```
for n, file in zip(num_k, files_em):
    path_actual = os.path.join(path, file)
    reconstruccion_fantoma = loadmat(path_actual)['EMr']
    mse_em.append(mse(fantoma_original, reconstruccion_fantoma))
    psnr_em.append(psnr(fantoma_original, reconstruccion_fantoma))
    ssim_em.append(ssim(fantoma_original, reconstruccion_fantoma, data_range=reconstruccion_fantoma.max()-reconstruccion_fantoma.min()))

theta = np.linspace(0., 180., 360, endpoint=False)
sinogram = radon(img_eq, theta=theta)
%timeit iradon(sinogram, theta=theta, filter_name='ramp')

#########################################################################

MATLAB

% Ejercicio 2 - ART, Simetric ART, Random ART, SART
close all

% Set the parameters for the test problem.

for step = [5, 3, 1, 0.5]
    N = 256;             % The discretization points.
    p = 360;             % No. of parallel rays.
    eta = 0.0001;        % Relative noise level.
    k = 100;             % No. of iterations.
    theta = 0:step:179;   % No. of used angles.
    % Create the test problem.
    [A,b_ex,x_ex] = paralleltomo(N,theta,p);
    % Noise level.
    delta = eta*norm(b_ex);
    % Add noise to the rhs.
    randn('state',0);
    e = randn(size(b_ex));
    e = delta*e/norm(e);
    b = b_ex + e;
    % Show the exact solution.
    figure
    imagesc(reshape(x_ex,N,N)), colormap gray,
    axis image off
    c = caxis;
    title('Exact phantom')
    fantoma = reshape(x_ex,N,N);
    %fantomaname = ['fantoma' num2str(step) '.mat'];
    %save(fantomaname, 'fantoma'); % Save the matrix to a .mat file
    % Perform the kaczmarz iterations.
    Xkacz = kaczmarz(A,b,k);
    % Show the kaczmarz solution.
    figure
    imagesc(reshape(Xkacz,N,N)), colormap gray,
```

```
    axis image off
    caxis(c);
    title('Kaczmarz reconstruction')
    ART = reshape(Xkacz,N,N);
    ARTname = ['ART_angles' num2str(step) '.mat'];
    save(ARTname, 'ART'); % Save the matrix to a .mat file
    % Perform the symmetric kaczmarz iterations.
    Xsymk = symkaczmarz(A,b,k);
    % Show the symmetric kaczmarz solution.
    figure
    imagesc(reshape(Xsymk,N,N)), colormap gray,
    axis image off
    caxis(c);
    title('Symmetric Kaczmarz reconstruction')
    symART = reshape(Xsymk,N,N);
    symARTname = ['symART_angles' num2str(step) '.mat'];
    save(symARTname, 'symART'); % Save the matrix to a .mat file
    % Perform the randomized kaczmarz iterations.
    Xrand = randkaczmarz(A,b,k);
    % Show the randomized kaczmarz solution.
    figure
    imagesc(reshape(Xrand,N,N)), colormap gray,
    axis image off
    caxis(c);
    title('Randomized Kaczmarz reconstruction')
    randART = reshape(Xrand,N,N);
    randARTname = ['randART_angles' num2str(step) '.mat'];
    save(randARTname, 'randART'); % Save the matrix to a .mat file
    % Perform SART
    Xsart = sart(A,b,k);
    % Show the SART solution.
    figure
    imagesc(reshape(Xsart,N,N)), colormap gray,
    axis image off
    caxis(c);
    title('SART reconstruction')
    SART = reshape(Xsart,N,N);
    SARTname = ['SART_angles' num2str(step) '.mat'];
    save(SARTname, 'SART');
end

for p = [30, 60, 90, 180, 360]
    N = 256;             % The discretization points
    %p = 360;            % No. of parallel rays.
    eta = 0.0001;       % Relative noise level.
    k = 100;            % No. of iterations.
    theta = 0:0.5:179;   % No. of used angles.
    % Create the test problem.
    [A,b_ex,x_ex] = paralleltomo(N,theta,p);
    % Noise level.
    delta = eta*norm(b_ex);
    % Add noise to the rhs.
```

```
randn('state',0);
e = randn(size(b_ex));
e = delta*e/norm(e);
b = b_ex + e;
% Show the exact solution.
figure
imagesc(reshape(x_ex,N,N)), colormap gray,
axis image off
c = caxis;
title('Exact phantom')
fantoma = reshape(x_ex,N,N);
%fantomaname = ['fantoma' num2str(step) '.mat'];
%save(fantomaname, 'fantoma'); % Save the matrix to a .mat file
% Perform the kaczmarz iterations.
Xkacz = kaczmarz(A,b,k);
% Show the kaczmarz solution.
figure
imagesc(reshape(Xkacz,N,N)), colormap gray,
axis image off
caxis(c);
title('Kaczmarz reconstruction')
ART = reshape(Xkacz,N,N);
ARTname = ['ART_detectorsv2' num2str(p) '.mat'];
save(ARTname, 'ART'); % Save the matrix to a .mat file
% Perform the symmetric kaczmarz iterations.
Xsymk = symkaczmarz(A,b,k);
% Show the symmetric kaczmarz solution.
figure
imagesc(reshape(Xsymk,N,N)), colormap gray,
axis image off
caxis(c);
title('Symmetric Kaczmarz reconstruction')
symART = reshape(Xsymk,N,N);
symARTname = ['symART_detectorsv2' num2str(p) '.mat'];
save(symARTname, 'symART'); % Save the matrix to a .mat file
% Perform the randomized kaczmarz iterations.
Xrand = randkaczmarz(A,b,k);
% Show the randomized kaczmarz solution.
figure
imagesc(reshape(Xrand,N,N)), colormap gray,
axis image off
caxis(c);
title('Randomized Kaczmarz reconstruction'
randART = reshape(Xrand,N,N);
randARTname = ['randART_detectorsv2' num2str(p) '.mat'];
save(randARTname, 'randART'); % Save the matrix to a .mat file
% Perform SART
Xsart = sart(A,b,k);
% Show the SART solution.
figure
imagesc(reshape(Xsart,N,N)), colormap gray,
axis image off
```

```
    caxis(c);
    title('SART reconstruction')
    SART = reshape(Xsart,N,N);
    SARTname = ['SART_detectorsv2' num2str(p) '.mat'];
    save(SARTname, 'SART');
end

for eta = [0.1, 0.01, 0.001, 0.0001, 0.00001]
    N = 256;
    p = 360;            % No. of parallel rays.
    k = 100;            % No. of iterations.
    theta = 0:0.5:179;   % No. of used angles.
    % Create the test problem.
    [A,b_ex,x_ex] = paralleltomo(N,theta,p);
    % Noise level.
    delta = eta*norm(b_ex);
    % Add noise to the rhs.
    randn('state',0);
    e = randn(size(b_ex));
    e = delta*e/norm(e);
    b = b_ex + e;
    % Show the exact solution.
    figure
    imagesc(reshape(x_ex,N,N)), colormap gray,
    axis image off
    c = caxis;
    title('Exact phantom')
    fantoma = reshape(x_ex,N,N);
    %fantomaname = ['fantoma' num2str(step) '.mat'];
    %save(fantomaname, 'fantoma'); % Save the matrix to a .mat file
    % Perform the kaczmarz iterations.
    Xkacz = kaczmarz(A,b,k);
    % Show the kaczmarz solution.
    figure
    imagesc(reshape(Xkacz,N,N)), colormap gray,
    axis image off
    caxis(c);
    title('Kaczmarz reconstruction')
    ART = reshape(Xkacz,N,N);
    ARTname = ['ART_rnoise' num2str(eta) '.mat'];
    save(ARTname, 'ART'); % Save the matrix to a .mat file
    % Perform the symmetric kaczmarz iterations.
    Xsymk = symkaczmarz(A,b,k);
    % Show the symmetric kaczmarz solution.
    figure
    imagesc(reshape(Xsymk,N,N)), colormap gray,
    axis image off
    caxis(c);
    title('Symmetric Kaczmarz reconstruction')
    symART = reshape(Xsymk,N,N);
    symARTname = ['symART_rnoise' num2str(eta) '.mat'];
    save(symARTname, 'symART'); % Save the matrix to a .mat file
```

```
    % Perform the randomized kaczmarz iterations.
    Xrand = randkaczmarz(A,b,k);
    % Show the randomized kaczmarz solution.
    figure
    imagesc(reshape(Xrand,N,N)), colormap gray,
    axis image off
    caxis(c);
    title('Randomized Kaczmarz reconstruction')
    randART = reshape(Xrand,N,N);
    randARTname = ['randART_rnoise' num2str(eta) '.mat'];
    save(randARTname, 'randART'); % Save the matrix to a .mat file
    % Perform SART
    Xsart = sart(A,b,k);
    % Show the SART solution.
    figure
    imagesc(reshape(Xsart,N,N)), colormap gray,
    axis image off
    caxis(c);
    title('SART reconstruction')
    SART = reshape(Xsart,N,N);
    SARTname = ['SART_rnoise' num2str(eta) '.mat'];
    save(SARTname, 'SART');
end


y = 10:10:1000;

N = 256;             % Tamano de imagen

p = 360;             % No. of parallel rays.

theta = 0:0.5:179;   % No. of used angles.

eta = 0.0001;          % Error relativo

% Create the test problem.
[A,b_ex,x_ex] = paralleltomo(N,theta,p);

% Noise level.
delta = eta*norm(b_ex);

% Add noise to the rhs.
randn('state',0);
e = randn(size(b_ex));
e = delta*e/norm(e);
b = b_ex + e;

for k = y
    % Perform Expectation Maximization
    XEM = em(A,b,k);
    figure
    imagesc(reshape(XEM,N,N)), colormap gray,
```

```
    axis image off
    caxis(c);
    title('Expectation maximization recontruct')
    EMr = reshape(XEM,N,N);
    EMname = ['EM_k' num2str(k) '.mat'];
    save(EMname, 'EMr');
end
```