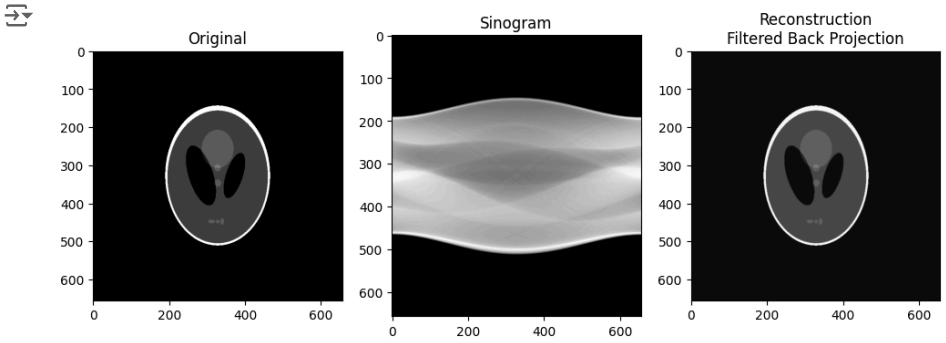
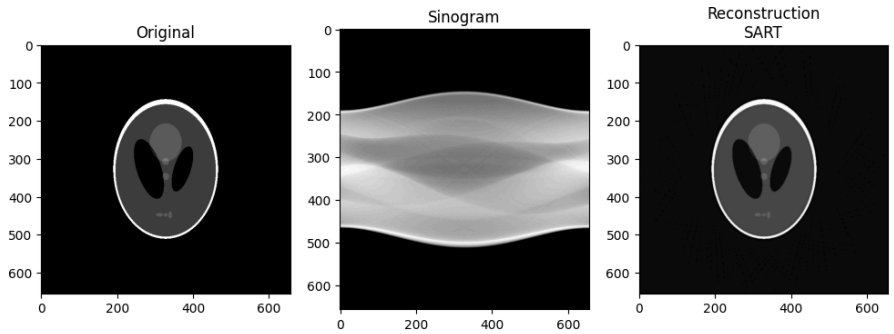
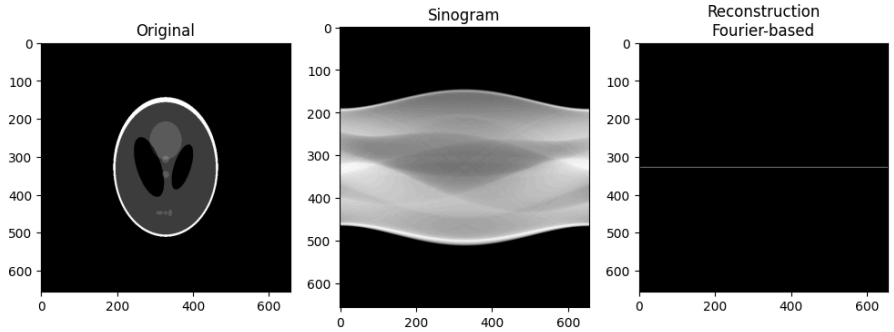
```
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import radon, iradon
from skimage.data import shepp_logan_phantom

# Create a Shepp-Logan phantom image
image = shepp_logan_phantom()
image = np.pad(image, ((128, 128), (128, 128)), 'constant')  # Padding for better visualization

# Compute the sinogram (Radon transform)
theta = np.linspace(0., 180., max(image.shape), endpoint=False)
sinogram = radon(image, theta=theta, circle=True)

# Perform the inverse Radon transform (Filtered Back Projection)
reconstruction_fbp = iradon(sinogram, theta=theta, filter_name='ramp')

# Plotting the results
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 4))
ax1.set_title("Original")
ax1.imshow(image, cmap=plt.cm.Greys_r)
ax2.set_title("Sinogram")
ax2.imshow(sinogram, cmap=plt.cm.Greys_r, aspect='auto')
ax3.set_title("Reconstruction\nFiltered Back Projection")
ax3.imshow(reconstruction_fbp, cmap=plt.cm.Greys_r)
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import radon, iradon_sart
from skimage.data import shepp_logan_phantom

# Create a Shepp-Logan phantom image
image = shepp_logan_phantom()
image = np.pad(image, ((128, 128), (128, 128)), 'constant')  # Padding for better visualization

# Compute the sinogram (Radon transform)
theta = np.linspace(0., 180., max(image.shape), endpoint=False)
sinogram = radon(image, theta=theta, circle=True)

# Perform the iterative reconstruction (SART)
reconstruction_sart = iradon_sart(sinogram, theta=theta)

# Plotting the results
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 4))
ax1.set_title("Original")
ax1.imshow(image, cmap=plt.cm.Greys_r)
ax2.set_title("Sinogram")
ax2.imshow(sinogram, cmap=plt.cm.Greys_r, aspect='auto')
ax3.set_title("Reconstruction\nSART")
ax3.imshow(reconstruction_sart, cmap=plt.cm.Greys_r)
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import radon
from numpy.fft import fft, ifft, fftshift, ifftshift

def fourier_reconstruction(sinogram, theta):
    # Compute the 1D Fourier transform of the projections
    projections_fft = fft(sinogram, axis=0)

    # Create the 2D Fourier space
    freq_x = np.fft.fftfreq(sinogram.shape[0])
    freq_y = np.fft.fftfreq(sinogram.shape[1])
    F = np.zeros((len(freq_x), len(freq_y)), dtype=complex)

    # Interpolate the 1D FFTs into the 2D Fourier space
    for i, angle in enumerate(theta):
        x = freq_x * np.cos(np.deg2rad(angle))
        y = freq_x * np.sin(np.deg2rad(angle))
        coords = np.vstack((y, x)).T
        F[np.round(coords[:, 0]).astype(int) + F.shape[0]//2, np.round(coords[:, 1]).astype(int) + F.shape[1]//2] = projections_fft[:, i]

    # Apply the inverse 2D Fourier transform
    reconstruction = np.abs(ifftshift(ifft(ifftshift(F))))
    return reconstruction

# Create a Shepp-Logan phantom image
image = shepp_logan_phantom()
image = np.pad(image, ((128, 128), (128, 128)), 'constant')  # Padding for better visualization

# Compute the sinogram (Radon transform)
theta = np.linspace(0., 180., max(image.shape), endpoint=False)
sinogram = radon(image, theta=theta, circle=True)

# Perform the Fourier-based reconstruction
reconstruction_fourier = fourier_reconstruction(sinogram, theta)

# Plotting the results
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 4))
ax1.set_title("Original")
ax1.imshow(image, cmap=plt.cm.Greys_r)
ax2.set_title("Sinogram")
ax2.imshow(sinogram, cmap=plt.cm.Greys_r, aspect='auto')
ax3.set_title("Reconstruction\nFourier-based")
ax3.imshow(np.abs(reconstruction_fourier), cmap=plt.cm.Greys_r)
plt.show()
```


Original
Sinogram
Reconstruction
Fourier-based
0
100
200
300
400
500
600
0
200
400
600

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.data import shepp_logan_phantom
from skimage.transform import radon, iradon

# Create a phantom image (Shepp-Logan phantom)
image = shepp_logan_phantom()
image = np.pad(image, ((100, 100), (100, 100)), mode='constant')

# Define the theta range for Radon transform
theta = np.linspace(0., 180., max(image.shape), endpoint=False)

# Analytical Reconstruction Method: Filtered Back Projection (FBP)
sinogram = radon(image, theta=theta, circle=True)
reconstructed_fbp = iradon(sinogram, theta=theta, filter_name='ramp')

# Iterative Reconstruction Method: Simple ART (Algebraic Reconstruction Technique)
def iterative_reconstruction(sinogram, theta, iterations=10):
    reconstructed = np.zeros((sinogram.shape[0], sinogram.shape[0]))
    for _ in range(iterations):
        for i, angle in enumerate(theta):
            projection = sinogram[:, i]
            projection = np.tile(projection, (reconstructed.shape[0], 1))
            rotation_matrix = np.rot90(projection, k=i)
            reconstructed += rotation_matrix
        reconstructed /= len(theta)
    return reconstructed

reconstructed_art = iterative_reconstruction(sinogram, theta, iterations=10)

# Fourier-based Reconstruction Method: Fourier Slice Theorem
def fourier_reconstruction(sinogram, theta):
    # Convert the sinogram to Fourier domain
    sinogram_ft = np.fft.fftshift(np.fft.fft2(sinogram))
    # Create an empty array in Fourier space
    reconstructed_ft = np.zeros_like(sinogram_ft, dtype=complex)
    # Populate the Fourier space according to the Fourier Slice Theorem
    for i, angle in enumerate(np.deg2rad(theta)):
        slice = np.fft.ifftshift(sinogram_ft[:, i])
        reconstructed_ft += np.outer(np.exp(-1j * angle * np.arange(sinogram.shape[0])), slice)
    # Inverse Fourier transform to get the image
    reconstructed = np.abs(np.fft.ifft2(np.fft.ifftshift(reconstructed_ft)))
    return reconstructed

reconstructed_fourier = fourier_reconstruction(sinogram, theta)

# Plot the results
fig, axes = plt.subplots(2, 2, figsize=(8, 8))
axes[0, 0].imshow(image, cmap=plt.cm.Greys_r)
axes[0, 0].set_title('Original Image')

axes[0, 1].imshow(reconstructed_fbp, cmap=plt.cm.Greys_r)
axes[0, 1].set_title('FBP Reconstruction')
```

+ Code + Text

Start coding or generate with AI.

```
axes[1, 1].imshow(reconstructed_fourier, cmap=plt.cm.Greys_r)
axes[1, 1].set_title('Fourier Reconstruction')

for ax in axes.ravel():
    ax.axis('off')

plt.tight_layout()
plt.show()
```