Tomograf

Katarzyna Jaromirska 155910 Łukasz Grobelny 155898

# Implementacja

Zaimplementowano model tomografu stożkowego. Do implementacji tomografu wykorzystano Jupyter notebook z interaktywnymi polami.

Do stworzenia aplikacji wykorzystano biblioteki:

* numpy (działania na macierzach, operacje matematyczne, etc.),
* skimage (odczyt obrazu i konwersja formatu wartości obrazu),
* IPython.display i ipywigdets (interaktywne UI),
* matplotlib (wyświetlanie obrazów/wykresów),
* pydicom (odczyt/zapis plików DICOM),
* glob i re (pobór ścieżek do zdjęć i plików DICOM).

# Główne funkcje programu

## Transformata Radona

Symulacja procesu projekcji promieni rentgenowskich przez obiekt z różnych kierunków. Tomograf wykonuje pełen obrót obrazu wokół jego środka. Uprzednio kształt obrazu zmieniany jest na kwadrat (w osobnej funkcji).

def radon\_transform(image, detectors = 180, detectors\_angular\_span = 180, step = 1):

    image\_center = (image.shape[0] // 2, image.shape[1] // 2)

    image\_radius = max(image\_center) \* np.sqrt(2)

    emiter\_angles = np.deg2rad(np.arange(0, 360, step))

    radon\_image = np.zeros((len(emiter\_angles), detectors), dtype=image.dtype)

    detectors\_angle = np.deg2rad(detectors\_angular\_span)

    detectors\_gap = detectors\_angle / (detectors - 1)

    for i, angle in enumerate(emiter\_angles):

        emiter\_pos = (int(image\_radius \* np.cos(angle) + image\_center[0]),

                      int(image\_radius \* np.sin(angle) + image\_center[1]))

        for j in range(detectors):

            detector\_pos = (int(image\_radius \* (np.cos(angle + np.pi - detectors\_angle / 2 + j \* detectors\_gap)) + image\_center[0]),

                            int(image\_radius \* (np.sin(angle + np.pi - detectors\_angle / 2 + j \* detectors\_gap)) + image\_center[1]))

            line\_pixels = bresenham(emiter\_pos[0], emiter\_pos[1], detector\_pos[0], detector\_pos[1])

            valid\_pixels = [(y, x) for y, x in line\_pixels if 0 <= y < image.shape[0] and 0 <= x < image.shape[1]]

            if valid\_pixels:

                radon\_image[i, j] = np.mean([image[y, x] for y, x in valid\_pixels])

    return radon\_image

## Odwrócona transformata Radona

Implementacja odwróconej transformaty Radona wraz z zapisem każdej iteracji zrekonstruowanego obrazu w celu późniejszego stworzenia animacji.

inverse\_image\_steps = []

def inverse\_radon\_transform(image, image\_size, detectors\_angular\_span = 180, step = 1):

    global inverse\_image\_steps

    inverse\_image\_steps = []

    reconstructed\_image = np.zeros((image\_size[0], image\_size[1]), dtype=np.float64)

    # get scans/detectors from image

    scans = image.shape[0]

    detectors = image.shape[1]

    image\_center = (image\_size[0] // 2, image\_size[1] // 2)

    image\_radius = max(image\_center) \* np.sqrt(2)

    emiter\_angles = np.deg2rad(np.arange(0, 360, step))

    detectors\_angle = np.deg2rad(detectors\_angular\_span)

    detectors\_gap = detectors\_angle / (detectors - 1)

    for i, angle in enumerate(emiter\_angles):

        emiter\_pos = (int(image\_radius \* np.cos(angle) + image\_center[0]),

                        int(image\_radius \* np.sin(angle) + image\_center[1]))

        for j in range(detectors):

            detector\_pos = (int(image\_radius \* (np.cos(angle + np.pi - detectors\_angle / 2 + j \* detectors\_gap)) + image\_center[0]),

                            int(image\_radius \* (np.sin(angle + np.pi - detectors\_angle / 2 + j \* detectors\_gap)) + image\_center[1]))

            line\_pixels = bresenham(emiter\_pos[0], emiter\_pos[1], detector\_pos[0], detector\_pos[1])

            valid\_pixels = [(y, x) for y, x in line\_pixels if 0 <= y < image\_size[0] and 0 <= x < image\_size[1]]

            if valid\_pixels:

                rows, cols = zip(\*valid\_pixels)

                reconstructed\_image[rows, cols] += image[i, j]

        inverse\_image\_steps.append(normalize\_image(reconstructed\_image))

    return reconstructed\_image

## Obliczanie jasności poszczególnych punktów obrazu wynikowego

W zaznaczonej linii obliczana zostaje jasność poszczególnych punktów obrazu. Na początku pobierane są pozycje pikseli z linii pomiędzy skanerem a detektorem, a następnie dodawana jest wartość jasności piksela [emiter, detector] z sinogramu na linii. Na końcu wynikowy obraz jest normalizowany na podstawie 99.9 percentyla.

    for i, angle in enumerate(emiter\_angles):

        emiter\_pos = (int(image\_radius \* np.cos(angle) + image\_center[0]),

                        int(image\_radius \* np.sin(angle) + image\_center[1]))

        for j in range(detectors):

            detector\_pos = (int(image\_radius \* (np.cos(angle + np.pi - detectors\_angle / 2 + j \* detectors\_gap)) + image\_center[0]),

                            int(image\_radius \* (np.sin(angle + np.pi - detectors\_angle / 2 + j \* detectors\_gap)) + image\_center[1]))

            line\_pixels = bresenham(emiter\_pos[0], emiter\_pos[1], detector\_pos[0], detector\_pos[1])

            valid\_pixels = [(y, x) for y, x in line\_pixels if 0 <= y < image\_size[0] and 0 <= x < image\_size[1]]

            if valid\_pixels:

                rows, cols = zip(\*valid\_pixels)

**reconstructed\_image[rows, cols] += image[i, j]**

## Interaktywny wybór parametrów tomografu

Użytkownik może wybrać: plik do przeprowadzenia symulacji tomografu (.jpg lub .dcm), liczbę detektorów, rozpiętość kątową detektorów, krok z jakim układ emiter-detektor poruszają się.

filenames = glob.glob(".\\images\\\*.jpg") + glob.glob(".\\dicom\\\*.dcm")

widgets.interact(set\_parameters,

         f=widgets.Dropdown(options=filenames, value='.\\images\\Kropka.jpg', description='Filename', layout=widgets.Layout(width='80%'), style={'description\_width': '200px'}),

         d=widgets.IntSlider(value=180, min=90, max=720, step=30, description='Amount of detectors:', layout=widgets.Layout(width='80%'), style={'description\_width': '200px'}),

         arch=widgets.IntSlider(value=180, min=90, max=360, step=10, description='Detectors angular span:', layout=widgets.Layout(width='80%'), style={'description\_width': '200px'}),

         i=widgets.IntSlider(value=2, min=1, max=10, step=1, description='Emiter-detectors rotate steps:', layout=widgets.Layout(width='80%'), style={'description\_width': '200px'})

)

widgets.interact(set\_is\_filter, val=widgets.Checkbox(value=True, description='Sinogram filtering'))

button = widgets.Button(description="Show results")

button.on\_click(show\_plot)

display(button)

## Pozyskiwanie odczytów poszczególnych detektorów

Wyznaczana jest linia pomiędzy położeniem skanu a detektorem na siatce pikseli. W tym celu wykorzystany został algorytm linii Bresenhama określający które piksele powinny zostać wybrane.

def bresenham(x0, y0, x1, y1):

    points = []

    dx = abs(x1 - x0)

    dy = abs(y1 - y0)

    sx = 1 if x0 < x1 else -1

    sy = 1 if y0 < y1 else -1

    err = dx - dy

    while True:

        points.append((y0, x0))

        if x0 == x1 and y0 == y1:

            break

        e2 = 2 \* err

        if e2 > -dy:

            err -= dy

            x0 += sx

        if e2 < dx:

            err += dx

            y0 += sy

    return points

## Filtrowanie sinogramu

Zaimplementowano funkcję filtrującą sinogram z zastosowaniem konwolucji. Wybrano standardową maskę o rozmiarze 21 elementów. Funkcja jest jednak   
w stanie operować na innych rozmiarach maski określanych w postaci parametru.

def filter\_sinogram(image, kernel\_size=21):

    # make it odd

    if kernel\_size % 2 == 0:

        kernel\_size += 1

    kernel = np.zeros(kernel\_size)

    kernel\_center = kernel\_size // 2

    kernel[kernel\_center] = 1

    for i in range(kernel\_center + 1, kernel\_size, 2):

        val = (-4 / np.pi\*\*2) / (i - kernel\_center)\*\*2

        kernel[i] = val

        kernel[-(i + 1)] = val

    filtered\_image = np.zeros((image.shape[0], image.shape[1]), dtype=image.dtype)

    for i in range(image.shape[0]):

        filtered\_image[i, :] = np.convolve(image[i, :], kernel, mode='same')

    return filtered\_image

## Wyznaczanie RMSE - pierwiastka błędu średniokwadratowego

def rmse(image\_in, image\_out):

    in\_img = img\_as\_float64(image\_in)

    out\_img = img\_as\_float64(image\_out)

    return (np.mean(in\_img - out\_img)\*\*2)\*\*0.5

## Odczyt i zapis plików DICOM

Odczyt plików polega na sprawdzeniu, czy wybrany przez użytkownika plik do przeprowadzenia symulacji tomografu jest plikiem DICOM. W przypadku pozytywnej weryfikacji, do odczytu tablicy pikseli obrazu wykorzystywana jest biblioteka **pydicom**.

    if re.match(r".\*.dcm", filename):

        dcm\_file = pydicom.dcmread(filename)

        in\_image = dcm\_file.pixel\_array.astype(np.ubyte)

        in\_image = img\_as\_ubyte(in\_image)

    else:

        in\_image = io.imread(filename)

        in\_image = img\_as\_ubyte(in\_image)

Do zapisu pliku DICOM użytkownik jest w stanie dodać następujące informacje: identyfikator pacjenta, imię i nazwisko, płeć, datę urodzenia, datę przeprowadzenia badania, zbadaną część ciała oraz komentarz. Istnieje również możliwość zmiany nazwy zapisanego pliku. Tworzony jest zestaw informacji pliku DICOM oraz dodawane są do niego podane przez użytkownika informacje. Generowane są wymagane identyfikatory dla obrazu tomograficznego, a sam obraz zostaje przekształcony do postaci bajtów. Na koniec plik zostaje zapisany.

def save\_DICOM(image, patient\_id: str, patient\_name: str, birth\_date, sex:str, study\_date, body\_part:str, comment="Check-up", filename="saved\_dicom"):

    filename += ".dcm"

    ds = pydicom.Dataset()

    ds.PatientName = patient\_name

    ds.PatientID = str(patient\_id)

    ds.PatientSex = 'M'

    ds.PatientBirthDate = birth\_date

    age = study\_date.year - birth\_date.year - ((study\_date.month, study\_date.day) < (birth\_date.month, birth\_date.day))

    ds.PatientAge = f"{age:03}Y"

    ds.BodyPartExamined = body\_part

    ds.Modality = "CT"

    ds.StudyDate = study\_date

    ds.StudyInstanceUID = pydicom.uid.generate\_uid()

    ds.SeriesInstanceUID = pydicom.uid.generate\_uid()

    ds.SOPInstanceUID = pydicom.uid.generate\_uid()

    ds.SOPClassUID = pydicom.uid.CTImageStorage

    ds.ImageType = ["ORIGINAL", "PRIMARY", "AXIAL"]

    ds.InstanceNumber = str(random.randint(1, 100))

    ds.ImagesInAcquisition = "1"

    ds.FrameOfReferenceUID = pydicom.uid.generate\_uid()

    ds.Rows, ds.Columns = image.shape

    ds.PhotometricInterpretation = "MONOCHROME2"

    ds.SamplesPerPixel = 1

    ds.BitsAllocated = 16

    ds.BitsStored = 16

    ds.HighBit = 15

    ds.PixelRepresentation = 0

    ds.PixelData = (image.astype(np.uint16)).tobytes() # to 16-bit

    file\_meta = pydicom.dataset.FileMetaDataset()

    file\_meta.MediaStorageSOPClassUID = pydicom.uid.SecondaryCaptureImageStorage

    file\_meta.MediaStorageSOPInstanceUID = pydicom.uid.generate\_uid()

    file\_meta.TransferSyntaxUID = pydicom.uid.ExplicitVRLittleEndian

    file\_meta.ImplementationClassUID = pydicom.uid.generate\_uid()

    file\_meta.ImplementationVersionName = "PYDICOM 2.0.0"

    ds.file\_meta = file\_meta

    ds.ImageComments = comment

    ds.is\_little\_endian = True

    ds.is\_implicit\_VR = False

    ds.save\_as(filename)

Stworzono interaktywny interfejs użytkownika do uzupełniania informacji   
o pacjencie. Użytkownik jest w stanie podać identyfikator pacjenta, jego imię   
i nazwisko, część ciała, płeć, a także ustawić datę urodzenia pacjenta, datę badania oraz skomentować zdjęcie i podać nazwę pliku. Zapisywany obraz tomograficzny to obraz wynikający z poprzednich interakcji użytkownika   
z interfejsem.

widgets.interact(set\_dicom\_params,

                 id = widgets.IntText(description="Patient ID:", value=1111111),

                 name = widgets.Text(description="Name and Surname:", value="ABC XYZ"),

                 sex = widgets.Dropdown(description="Sex:", options=['M', 'F'], value='M'),

                 birth = widgets.DatePicker(description="Birth date:", value=datetime.date.today()),

                 exam = widgets.DatePicker(description="Study date:", value=datetime.date.today()),

                 body\_part = widgets.Text(description="Body part:", value="HEAD"),

                 comment = widgets.Text(description="Comment:", value="Check-up"),

                 filename = widgets.Text(description="Filename (without extension)", value="saved\_dicom"))

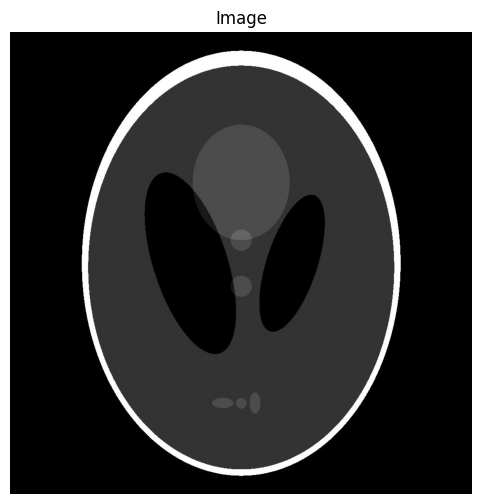
button = widgets.Button(description="Save as DICOM")

button.on\_click(on\_save\_cliked)

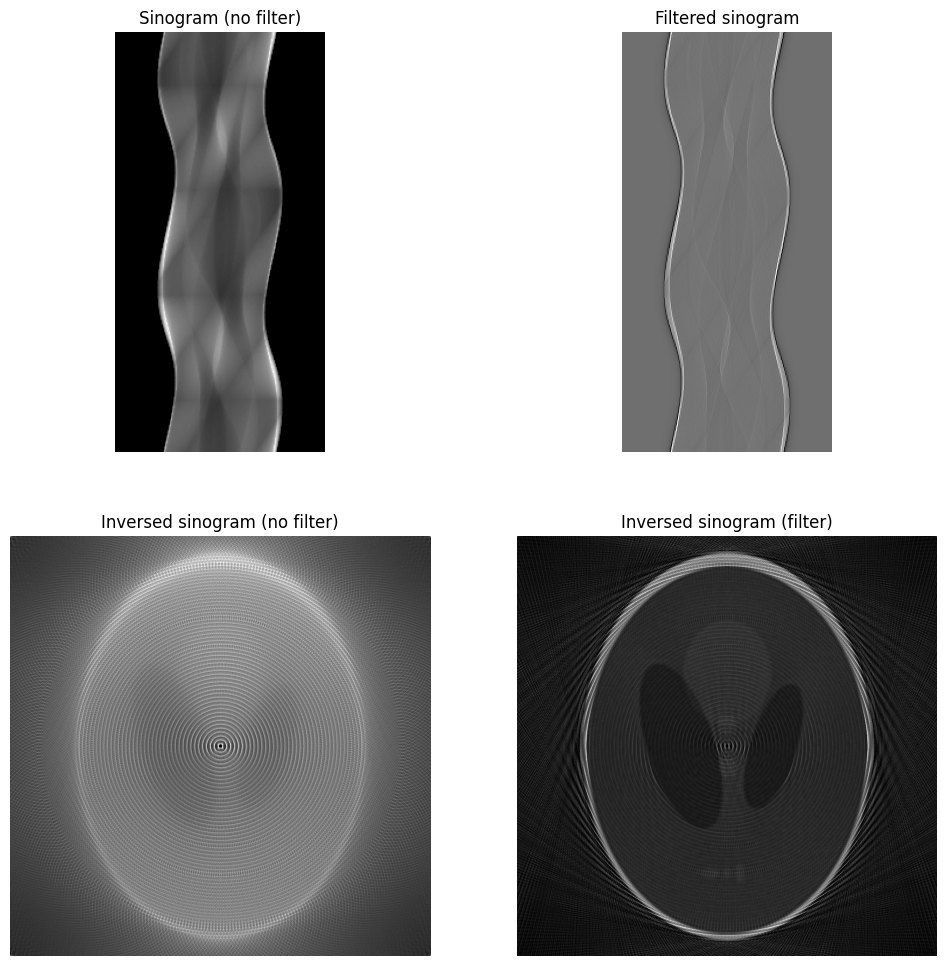
display(button)

# Działanie programu

Obraz wejściowy:

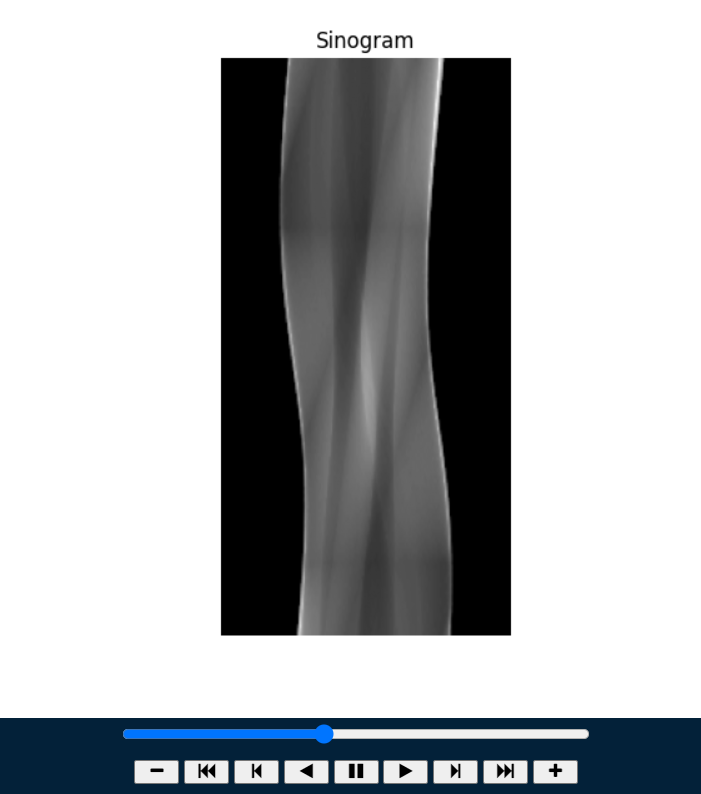


Obraz wynikowy:



Można zauważyć, że obrazy wynikowe posiadają lekkie artefakty w postaci „okręgów” i „pasków”.

Animacja tworzenia sinogramu (obraz sinogramu został rozciągnięty przez matplotlib):



Animacja rekonstrukcji obrazu:



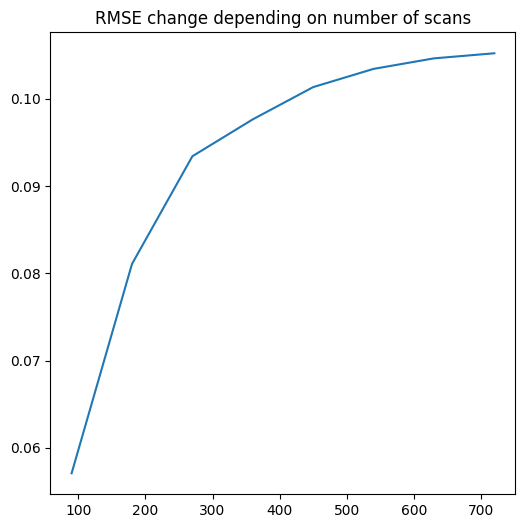
# Analiza wpływu parametrów na jakość obrazu wynikowego

### Podstawowe parametry:

* Ilość detektorów – 180
* Ilość skanów – 180
* Rozpiętość wachlarza detektorów - 180°

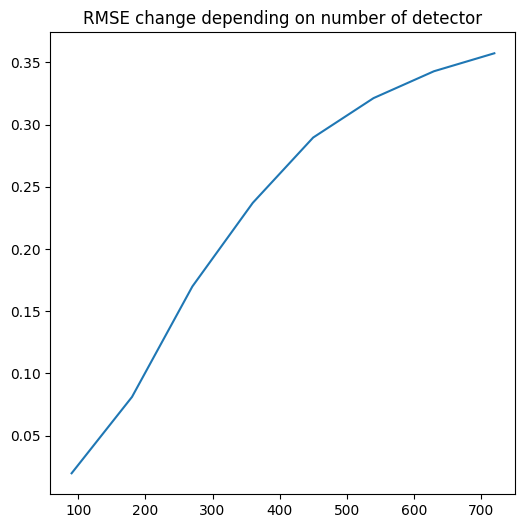
### Analiza ilości skanów

Wraz ze wzrostem ilości skanów maleje krok przesunięcia układu emiter-detektory. Może to prowadzić do nasycenia wartości na niektórych pikselach linii, co w konsekwencji może wpływać na wartość pierwiastka błędu średniokwadratowego (RMSE). Należy jednak zauważyć, że w przeprowadzonej analizie zmiana parametru liczby skanów powoduje jedynie niewielkie różnice w wartości RMSE.



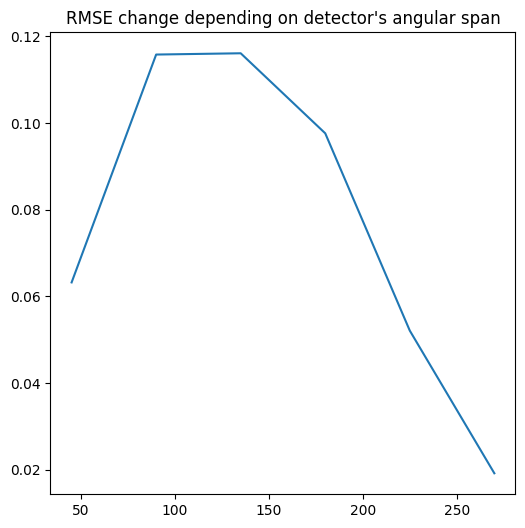
### Analiza ilości detektorów

Ze względu na rozpiętość wachlarza wynoszącą 180°, wiele detektorów może posiadać zbliżone linie emiter–detektor. Z uwagi na obserwowane w obrazach artefakty („paski”), podobne trajektowie mogą nasilać ten efekt, co w rezultacie może prowadzić do pogorszenia jakości odtworzonego obrazu.



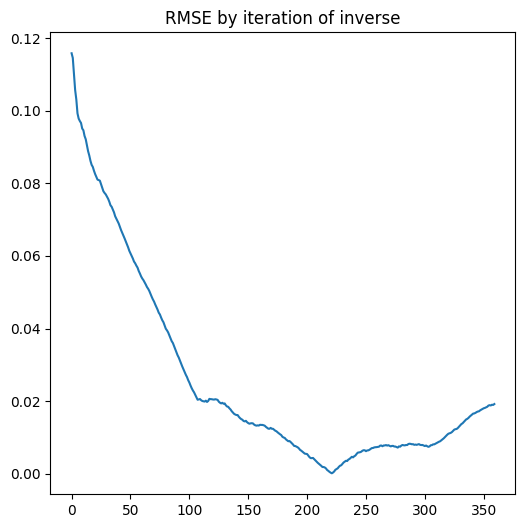
### Analiza rozpiętości wachlarza detektorów

Większość wyników jest zgodna z przewidywaniami. Im szersza rozpiętość stożka, tym detektory okazały się skuteczniejsze w skanowaniu obrazu. Zaskakuje jednak mały błąd uzyskany przy rozpiętości 45°, który okazał się nieoczekiwanie niski.



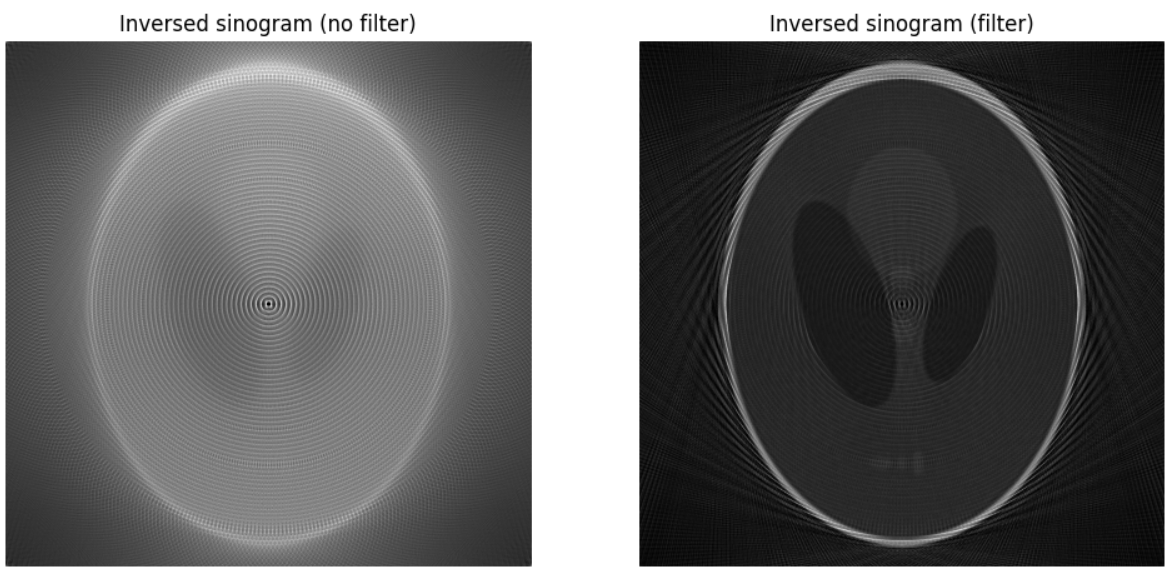
### Zmiana RMSE po iteracjach

Stopniowe rekonstruowanie obrazu zmniejsza błąd RMSE. Wyjątkiem jest jednak moment około 220 iteracji, gdzie błąd zaczyna stopniowo rosnąć. Może to być efektem pojawiających się „pasków” na rekonstruowanych obrazach, które wpływają na jakość rekonstrukcji.



### RMSE ze względu na użycie filtrowania

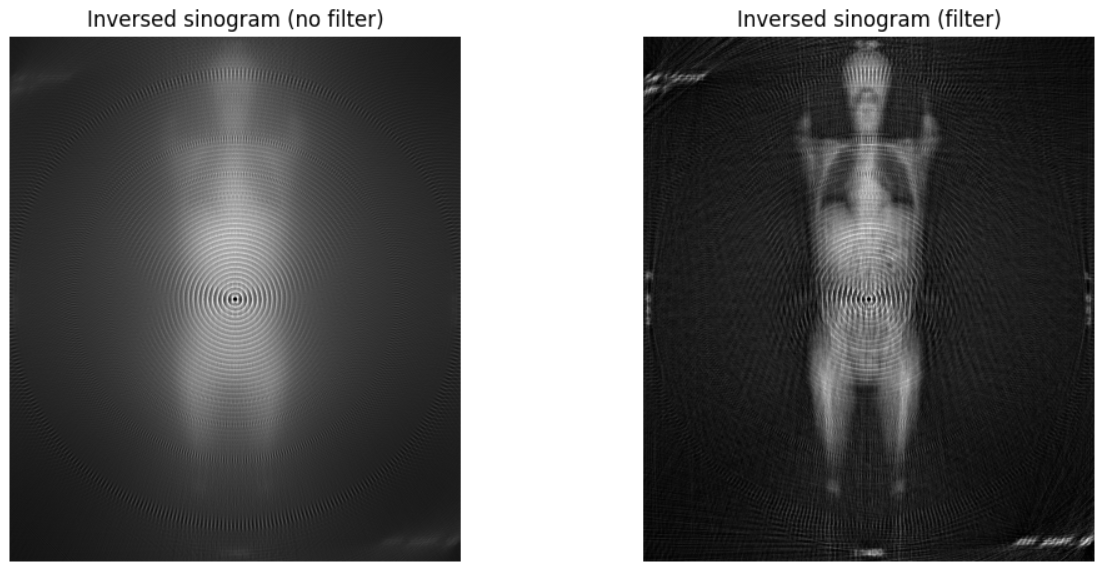
Dla obrazu Shepp\_logan.jpg



Błąd średniokwadratowy (bez filtra) - 0.303005

Błąd średniokwadratowy (filtr) - 0.019191

Dla obrazu CT\_ScoutView.jpg:



Błąd średniokwadratowy (bez filtra) - 0.172123

Błąd średniokwadratowy (filtr) - 0.080801