Tomograf

Katarzyna Jaromirska 155910 Łukasz Grobelny 155898

# Implementacja

Zaimplementowany model tomografu to stożkowy. Do zaimplementowania tomografu stworzono 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:

Implementacja stożkowego modelu tomografu. Tomograf robi pełny obrót wokół środka obrazu. Zmieniany jest również uprzednio kształt obrazu 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:

Odwrócona transformata również zapisuje każdą iterację rekonstrukcji obrazu do 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 podkreślonej linii obliczana zostaje jasność poszczególnych punktów. Najpierw pobierane są pozycje pikseli z linii pomiędzy skanerem a detektorem. Następnie dodawana jest wartość jasności piksela [emiter, detector] z sinogramu na linii.

    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]**

Wynikowy obraz jest również pod koniec normalizowany na podstawie 99.9 percentyla

## Interaktywny wybór parametrów tomografu:

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

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

Poprowadzona jest linia pomiędzy położeniem skanu a detektorem. Użyty został algorytm linii Bresenhama w celu znalezienia pikseli na tej linii.

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

Zaimplementowana została funkcja filtrująca sinogram z zastosowaniem konwolucji. Wybraliśmy standardową maskę wielkości 21. Funkcja działa też na inne rozmiary.

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 pliku DICOM. Jeśli tak, to użyta zostaje biblioteka pydicom do odczytania tablicy pikseli obrazu

    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 można dodać następujące informacje: ID pacjenta, imię i nazwisko, płeć, datę urodzenia, data przeprowadzenia badania, zbadaną część ciała, komentarz. Można też zmienić nazwę zapisanego pliku. Tworzony jest zestaw informacji pliku DICOM. Następnie dodawane są tam podane przez użytkownika informację, generowane są potrzebne identyfikatory do obrazu z tomografu, a obraz zostaje skonwertowany do bajtów. Pod koniec plik jest zapisywany.

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)

Stworzone zostało również interaktywne UI do uzupełnienia informacji o pacjencie. Użytkownik może podać ID pacjenta, imię i nazwisko, część ciała, wybrać płeć, ustawić datę urodzenia pacjenta, datę badania oraz skomentować zdjęcie i podać nazwę pliku do zapisania. Obraz z tomografu, jaki jest zapisywany, to obraz wynikowy z poprzednich interakcji użytkownika z UI.

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"))

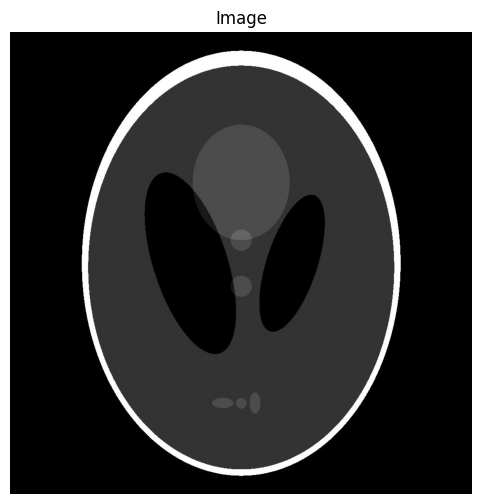
button = widgets.Button(description="Save as DICOM")

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

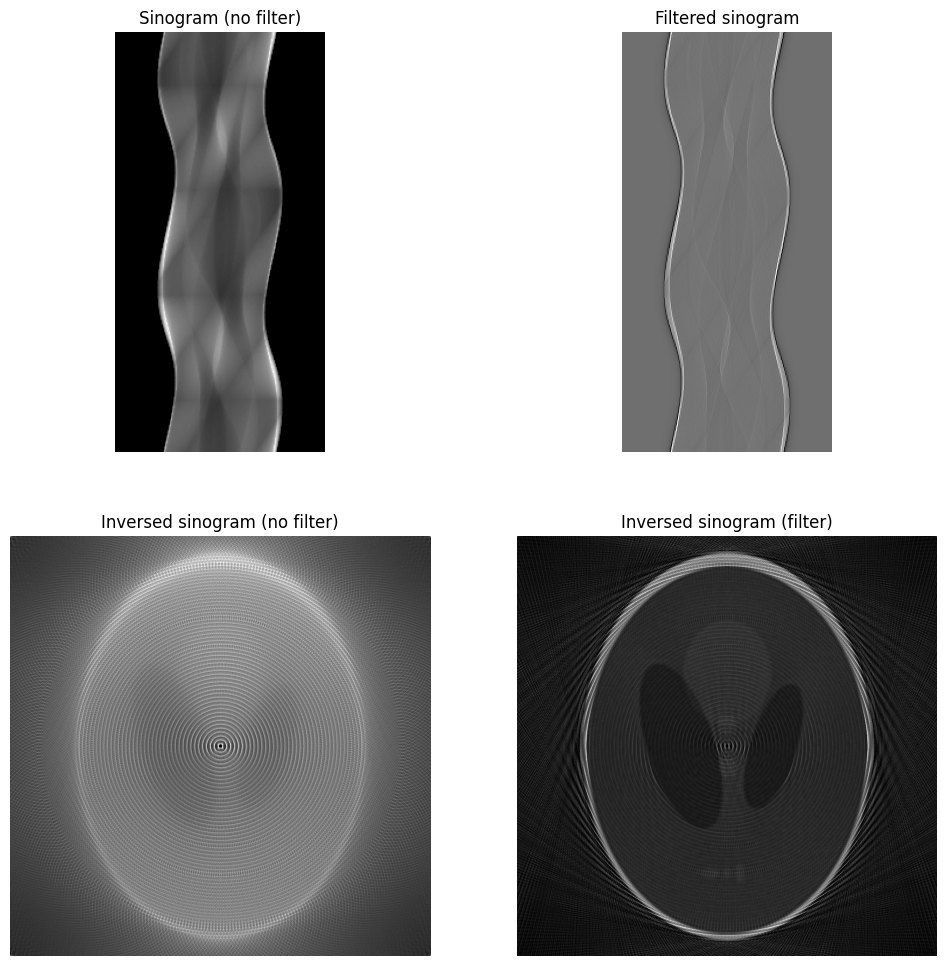
display(button)

# Działanie programu

Obraz wejściowy:

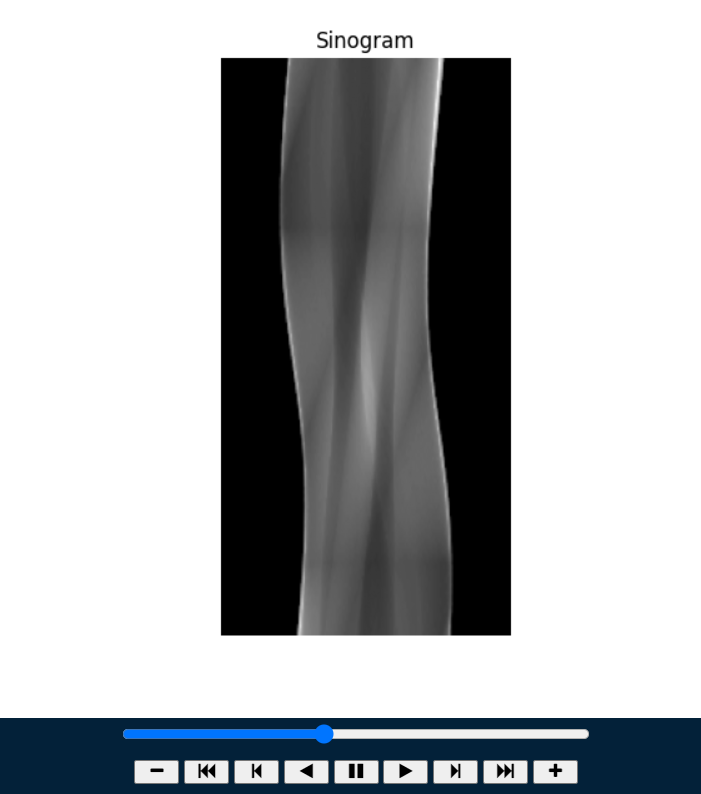


Wyniki:



Można zauważyć, że obrazy wynikowe mają lekkie „okręgi” i „paski”.

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



Animacja rekonstrukcji obrazu:



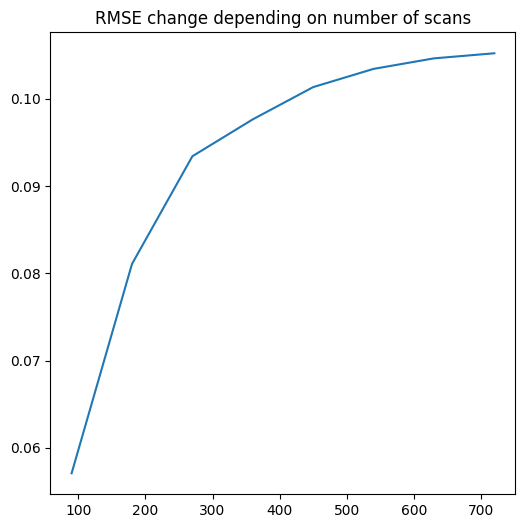
# Analiza wpływu parametrów na jakość obrazu wynikowego

### Podstawowe parametry:

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

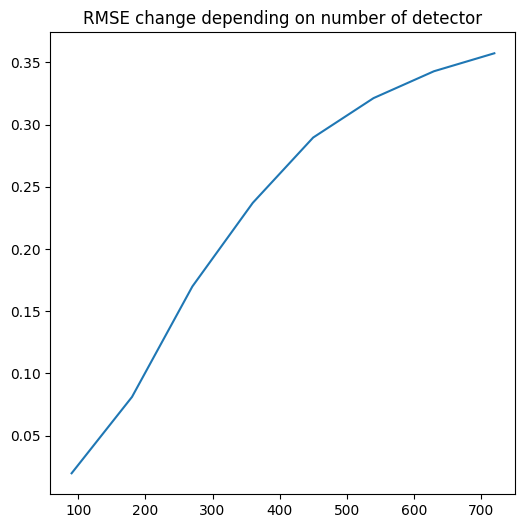
### Analiza ilości skanów

Im większa ilość skanów tym mniejszy krok przesunięcia układu emiter-detektory. Może to skutkować nasyceniem się na niektórych pikselach linii. Dlatego też może być tak, że wartość pierwiastka błędu średniokwadratowego. Jednakże warto zauważyć, iż w tej analizie zmiana parametru ilości skanów przynosi małą różnicę w RMSE.



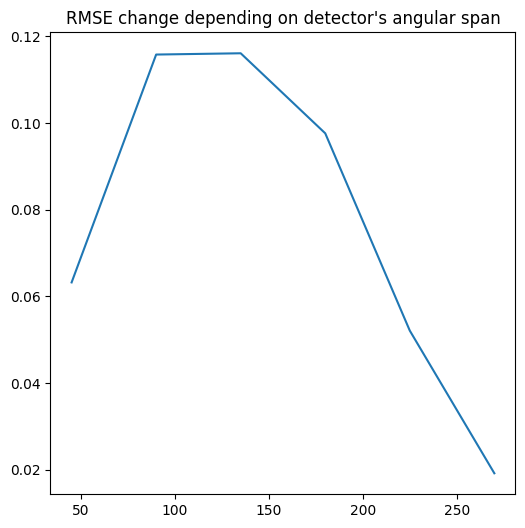
### Analiza ilości detektorów

Następnie sprawdzony został wpływ ilości detektorów na błąd RMSE. Ponieważ rozpiętość wachlarza to 180°, na wachlarzu będzie dużo detektorów o podobnych liniach emiter-detektor. Ponieważ obrazy wynikowe są lekko „paskowane”, może to skutkować nasileniem się tych pasków, pogarszając jakość obrazu.



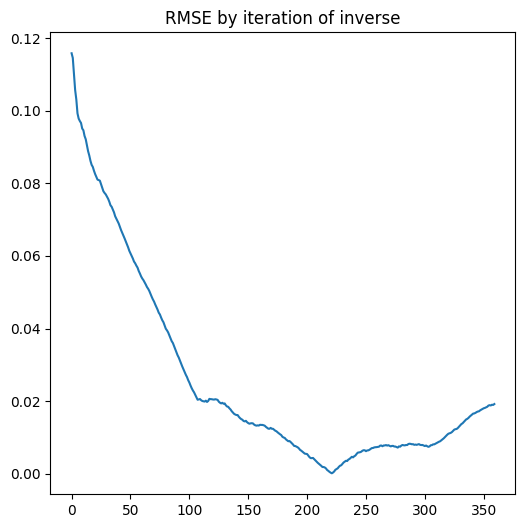
### Analiza rozpiętości wachlarza detektorów

Większość wyniku była do przewidzenia. Im bardziej rozpięty stożek, tym lepiej detektory przeskanują. Co dziwi to błąd przy rozpiętośći 45°, jest zaskakująco mały.



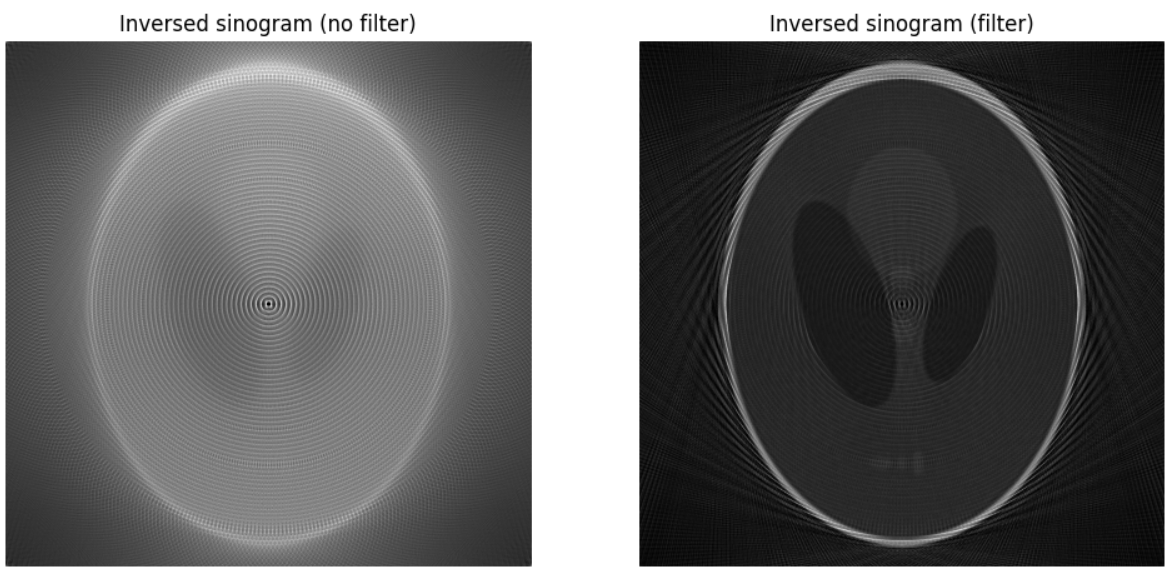
### Zmiana RMSE po iteracjach

Jak łatwo zauważyć, stopniowe rekonstruowanie obrazu zmniejsza błąd RMSE. Wyjątek natomiast jest od około 220 iteracji, gdzie tego błędu zaczyna przybywać. Powodem tego może być efekt „pasków”, które się tworzą na rekonstruowanych obrazach.



### RMSE ze względu na użycie filtrowania

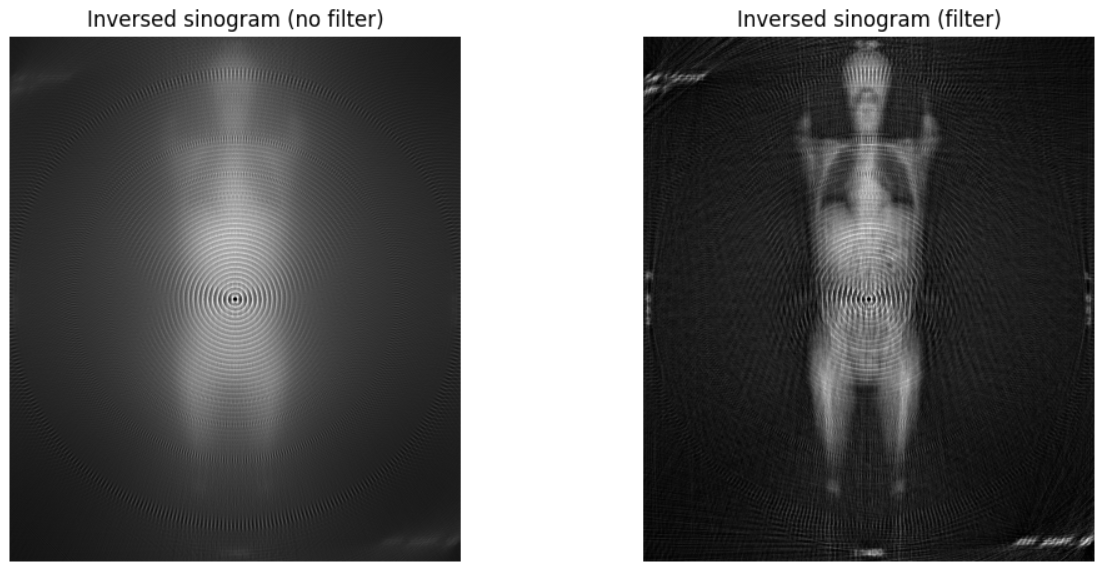
Dla obrazu Shepp\_logan.jpg



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