Instytut Informatyki
Wydział Informatyki i Telekomunikacji, Politechnika Poznańska

Laboratorium z przedmiotu Informatyka w Medycynie (IWM)			
Projekt nr 1			
Temat: Tomograf			
Prowadzący: *mgr inż. Iwo Błądek*	*Autor* Tobiasz Gruszczyński 145333 Sebastian Grabowski 145248	*Grupa dziekańska:*	*L15*
		Ocena:	

Zastosowany model tomografu

Przy wykonywaniu projektu wykorzystaliśmy model stożkowy.

Zastosowany język programowania

Zaimplementowaliśmy działanie tomografu w języku Python oraz wykorzystaliśmy Jupyther Notebook

Wykorzystane biblioteki:

```
import ipywidgets as widgets
import numpy as np
import matplotlib.pyplot as plt
import pydicom
import math
import IPython.display
import dateutil.parser
import datetime
import scipy.signal as sig
import re
import time
import cv2
import skimage.transform
import skimage.io
```

Rys.1 Biblioteki

Poznań 2022

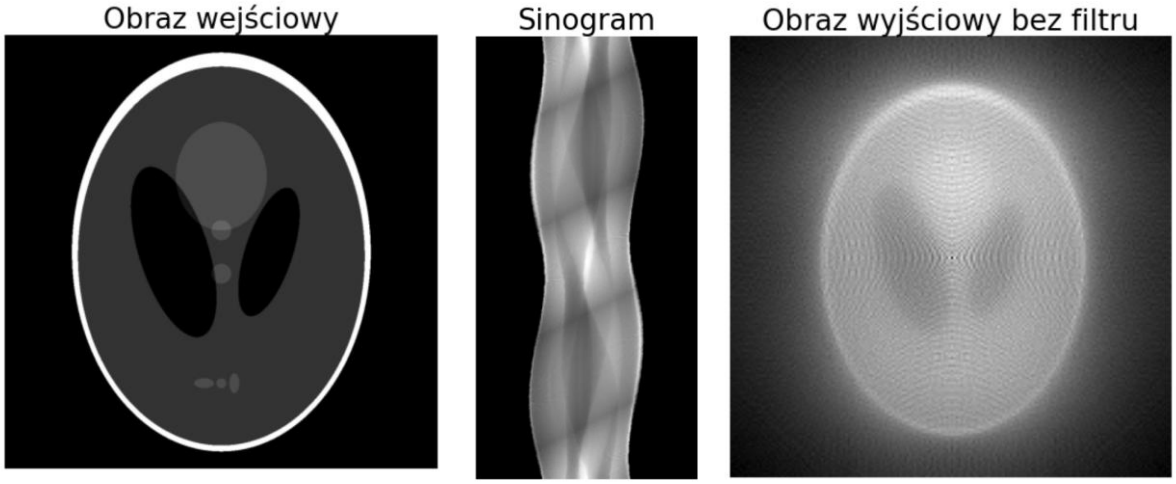
Opis głównych funkcji programu

Pozyskiwanie odczytów dla poszczególnych detektorów:

```python
# Definicja klas
class Sinogram:
    def __init__(self):
        self.sinograms = []

    def makeSinogram(self, img, iterations, step, detector_num, detector_rng):
        pos = EmitersDetectors.Positions()
        radius = round(len(img) / 2) - 1

        emiters, detectors = pos.setPositions(iterations, step, radius, detector_num, detector_rng)
        sinogram = np.zeros([len(emiters), len(detectors[0])])

        for i, [eX, eY] in enumerate(emiters):
            for j, [dX, dY] in enumerate(detectors[i]):
                line = Bresenham.BresenhamLine(eX, eY, dX, dY)

                for x, y in line:
                    sinogram[i, j] += img[x, y]
            self.sinograms.append(np.copy(sinogram))
```

Rys.2 Tworzenie sinogramu

Powyższa funkcja odpowiada za rozmieszczenie emiterów i detektorów. Następnie podczas kolejnych iteracji pętli iterujemy po wszystkich emiterach i dla każdego emitera po wszystkich detektorach. Kolejnym korkiem jest wykorzystanie algorytmu Bresehmana, z którego otrzymujemy linie sinogramu, która dodajemy do tablicy wyjściowej. Analogicznie odbywa się przetwarzanie sinogramu na obraz wyjściowy.



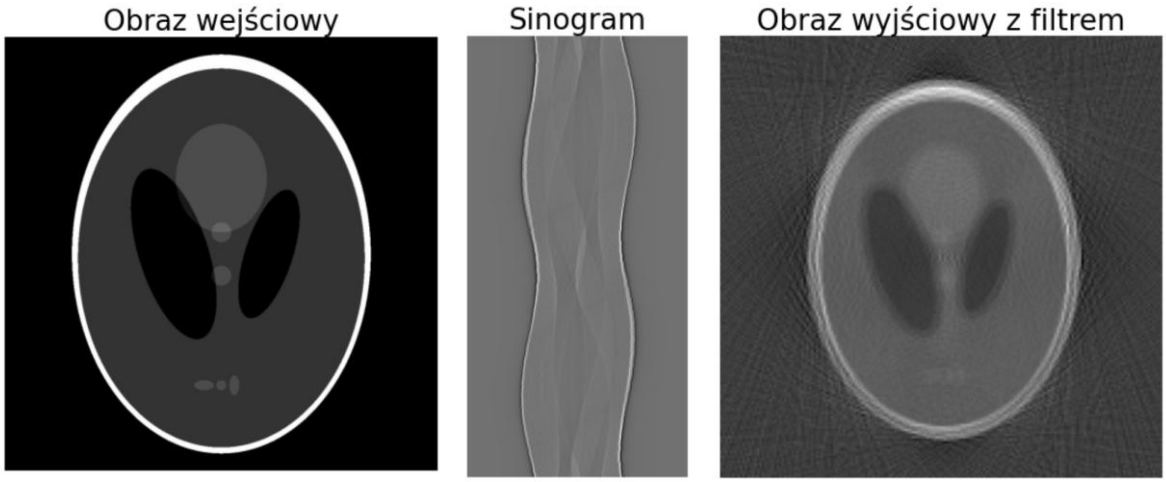
Rys.3 Przykładowy wynik działania funkcji

Filtrowanie sinogramu, zastosowany rozmiar maski:

```python
def do_mask(detectors):
    mask_size = floor(detectors / 2)
    mask = np.zeros(mask_size)
    center = floor(mask_size / 2)
    for i in range(0, mask_size, 1):
        k = i - center
        if k % 2 != 0:
            mask[i] = (-4 / np.pi ** 2) / k ** 2
    mask[center] = 1
    return mask


def filtering_sinogram(sinogram):
    sinogram_shape = np.shape(sinogram)
    number_of_projections = sinogram_shape[0]
    number_of_detectors = sinogram_shape[1]

    filtered = np.zeros((number_of_projections, number_of_detectors))
    mask = do_mask(number_of_detectors)

    # splot każdej projekcji z maską
    for projection in range(number_of_projections):
        filtered[projection] = sig.convolve(sinogram[projection], mask, mode='same', method='direct')

    return filtered
```

Rys.4 Poprawione filtrowanie sinogramu

Ten fragment kodu odpowiada za filtrowanie sinogramu. Najpierw odczytujemy liczbę projekcji oraz detektorów. Następnie w zależności od liczby detektorów dobieramy maskę (funkcja ***do_mask***). Gdy już maska zostanie obliczona, to dla każdej projekcji odbywa się jej splot z maską.



Rys.5 Przykładowy wynik filtrowania

Ustalenie jasności poszczególnych punktów obrazu oraz jego przetwarzanie końcowe (np. uśrednianie, normalizacja)

```
# Zdefiniowanie funkcji
def normalize(image):
    min_value = np.min(image)

    max_value = np.max(image)

    for i in range(len(image)):
        for j in range(len(image[i])):
            image[i, j] = (image[i, j] - min_value) / (max_value - min_value)

    return image
```

Rys.6 Normalizacja

Zadaniem funkcji ***normalize*** jest normalizacja otrzymanych wyników sinogramu oraz odwróconej transformaty sinogramu. Z kolei w celu uśrednienia obrazy wykorzystaliśmy poniższy fragment kodu.

```
self.img[radius - round(len(resized) / 2): radius + round(len(resized) / 2),
         radius - round(len(resized[0]) / 2): radius + round(len(resized[0]) / 2)] = resized
```

Rys.7 Uśrednianie

Wyznaczanie wartości miary RMSE na podstawie obrazu wejściowego oraz wyjściowego

```
def rmse(x, y):
    MSE = np.square(np.subtract(x, y)).mean()
    RMSE = math.sqrt(MSE)
    return RMSE
```

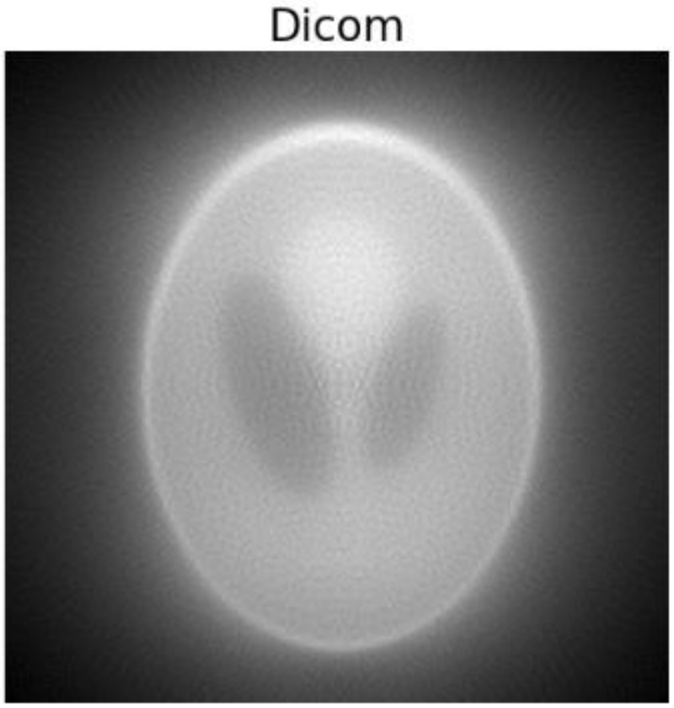
Rys.8 Uśrednianie

Funkcja ***rmse*** jest odpowiedzialna za obliczenie błędu średniokwadratowego. Najpierw odbywa się obliczenie MSE, czyli uśredniona suma potęgi różnic, a na końcu pierwiastkujemy poprzedni wynik otrzymując RMSE.

Odczyt i zapis plików DICOM

```
def writeDicom(image, name, comment):
    filename = get_testdata_files("CT_small.dcm")[0]
    ds = pydicom.dcmread(filename)

    def normalizeInDicom(image_temp):
        maximum = 0
        for vector in image_temp:
            if max(vector) > maximum:
                maximum = max(vector)
        for i in range(len(image_temp)):
            for x in range(len(image_temp[0])):
                if maximum != 0 and image_temp[i][x] > 0:
                    image_temp[i][x] = image_temp[i][x]*1024/maximum
                else:
                    image_temp[i][x] = 0
        return image_temp

    image = normalizeInDicom(image)
    image2 = np.asarray(image, dtype=np.uint16)
    ds.Rows = image2.shape[1]
    ds.Columns = image2.shape[0]
    ds.PixelData = image2.tostring()
    ds.PatientName = name
    ds.InstitutionName = 'Politechnika Poznanska'
    ds.Manufacturer = 'Politechnika Poznanska'
    dt = datetime.datetime.now()
    ds.StudyDate = dt.strftime('%Y%m%d')
    timeStr = dt.strftime('%H%M%S.%d')
    ds.StudyTime = timeStr
    ds.AdditionalPatientHistory = comment
    ds.save_as("Dicom_File.dcm")
```

```
if tom.isDicom:
    Dicom.writeDicom(np.copy(reverse), tom.patient, tom.descript)
    reverseD = pydicom.dcmread("Dicom_File.dcm")
    tom.dicom = reverseD
```



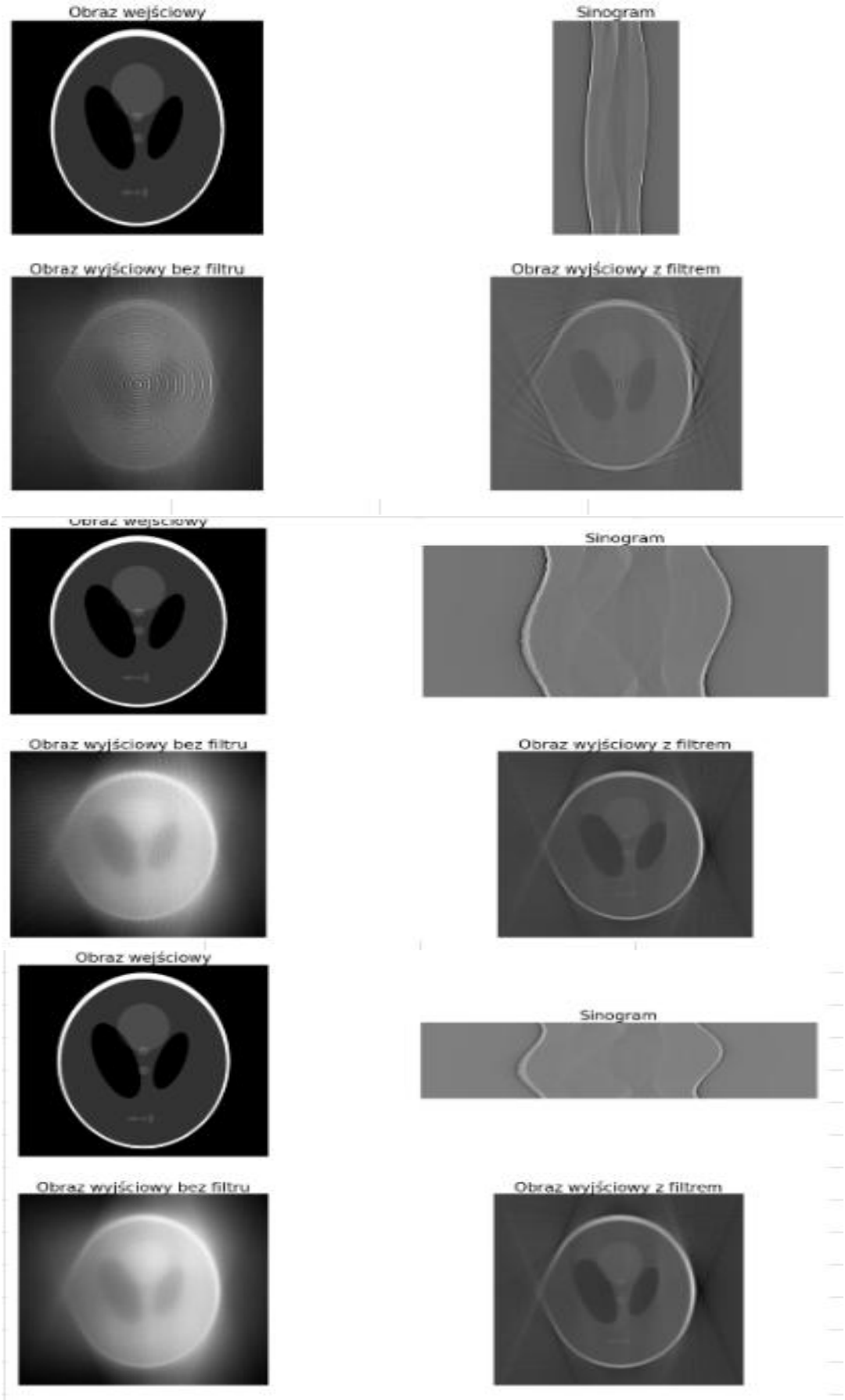
Funkcja writeDicom jest odpowiedzialna za obraz Dicom, powyżej znajduje się przykladowy obraz odczytany z pliku .dcm

Wynik eksperymentu sprawdzającego wpływ poszczególnych parametrów (liczba detektorów, liczba skanów, rozpiętość stożka/wachlarza z detektorami) na jakość obrazu wynikowego wyrażoną za pomocą miary RMSE. Jako wartości domyślne proszę przyjąć 180 detektorów, 180 skanów oraz rozpiętość wachlarza równą 180 stopni (PI)

Liczba detektorów zmienia się od 90 do 720 z krokiem 90

Detektory		
Liczba Detektorów	***Wyniki MSE (bez filtru)***	***Wyniki MSE (z filtrem)***
90	0.25	46.77
180	0.35	41.40
270	0.41	43.57
360	0.38	41.17
450	0.39	44.65
540	0.39	41.80
630	0.38	44.51
720	0.38	41.59

Poniższe zdjęcie to odpowiadają kolejno 90, 360 oraz 720 detektorom:

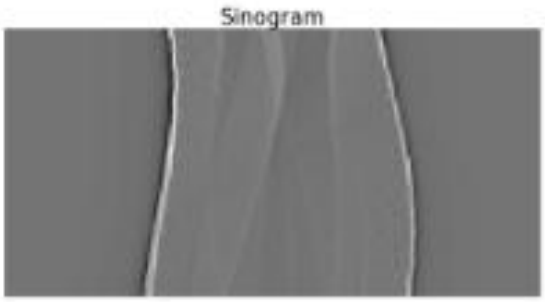


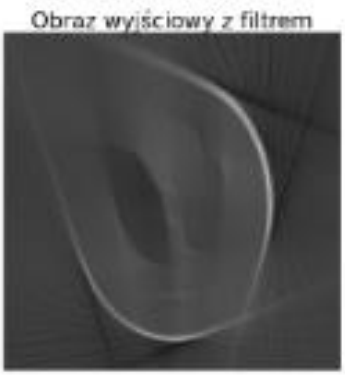
Wraz ze wzrostem liczby detektorów sinogram robi się coraz szerszy, a obraz bardziej wyraźny.

Liczba skanów(iteracji) zmienia się od 90 do 720 z krokiem 90

Skany		
Liczba iteracji	***Wyniki MSE (bez filtru)***	***Wyniki MSE (z filtrem)***
90	0.40	25.67
180	0.35	41.40
270	0.38	57.63
360	0.37	70.66
450	0.37	89.75
540	0.36	108.38
630	0.38	126.56
720	0.37	141.42

Poniższe zdjęcie to odpowiadają kolejno 90, 360 oraz 720 iteracjom:



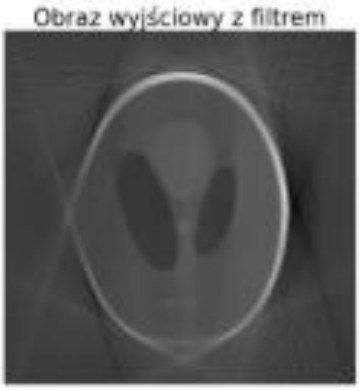


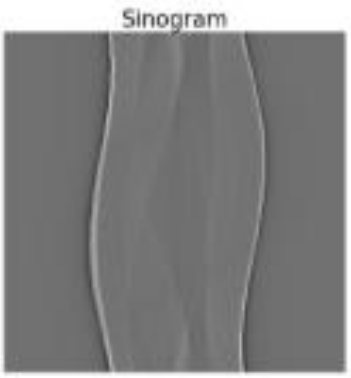
Wraz ze wzrostem liczby iteracji sinogram się wydłuża. Dzieje się tak za sprawą ilości próbek wykonanych przez pary emiter-detekro. Większa ilość próbek powoduje poprawe jakości obrazu.

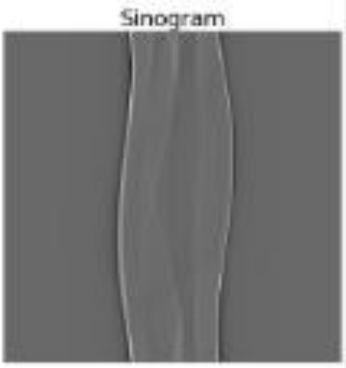
Rozpiętość wachlarza zmienia się od 45 do 270 z krokiem 45 stopni

Rozpiętość wachlarza		
Rozpiętość wachlarza	***Wyniki MSE (bez filtru)***	***Wyniki MSE (z filtrem)***
45	0.30	71.31
90	0.37	52.08
135	0.41	42.88
180	0.35	41.40
225	0.34	41.68
270	0.37	42.17

Poniższe zdjęcie to odpowiadają kolejno 90, 180 oraz 270 stopniom.



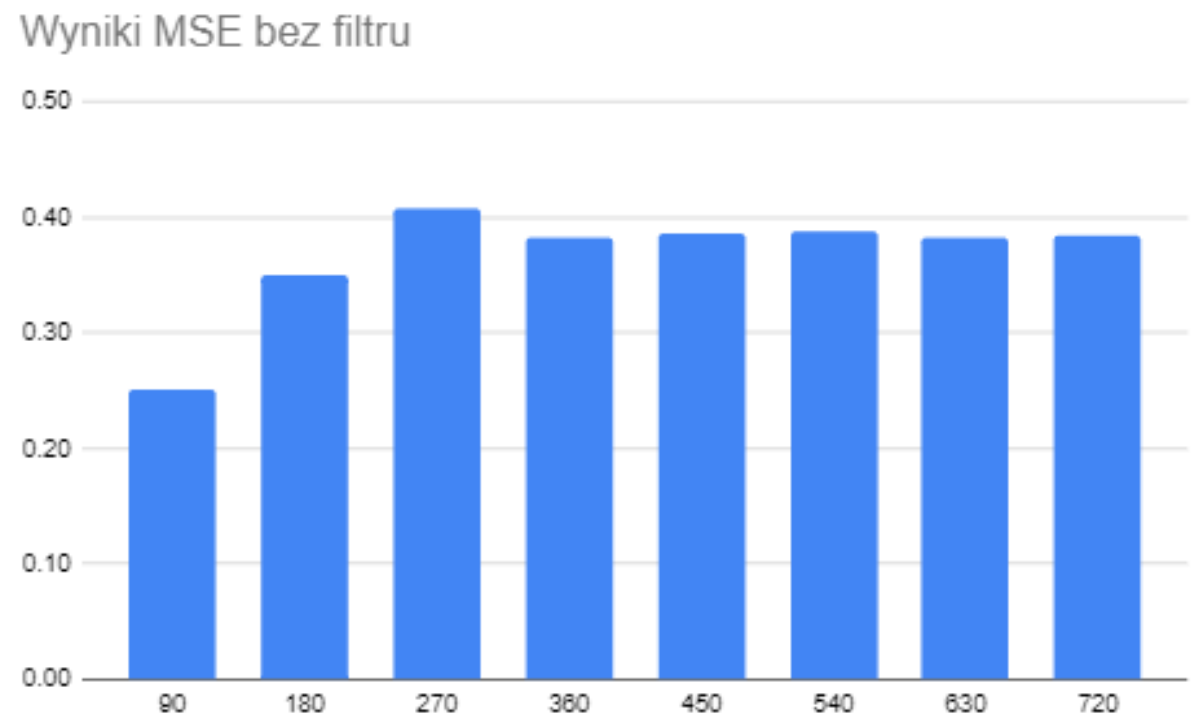




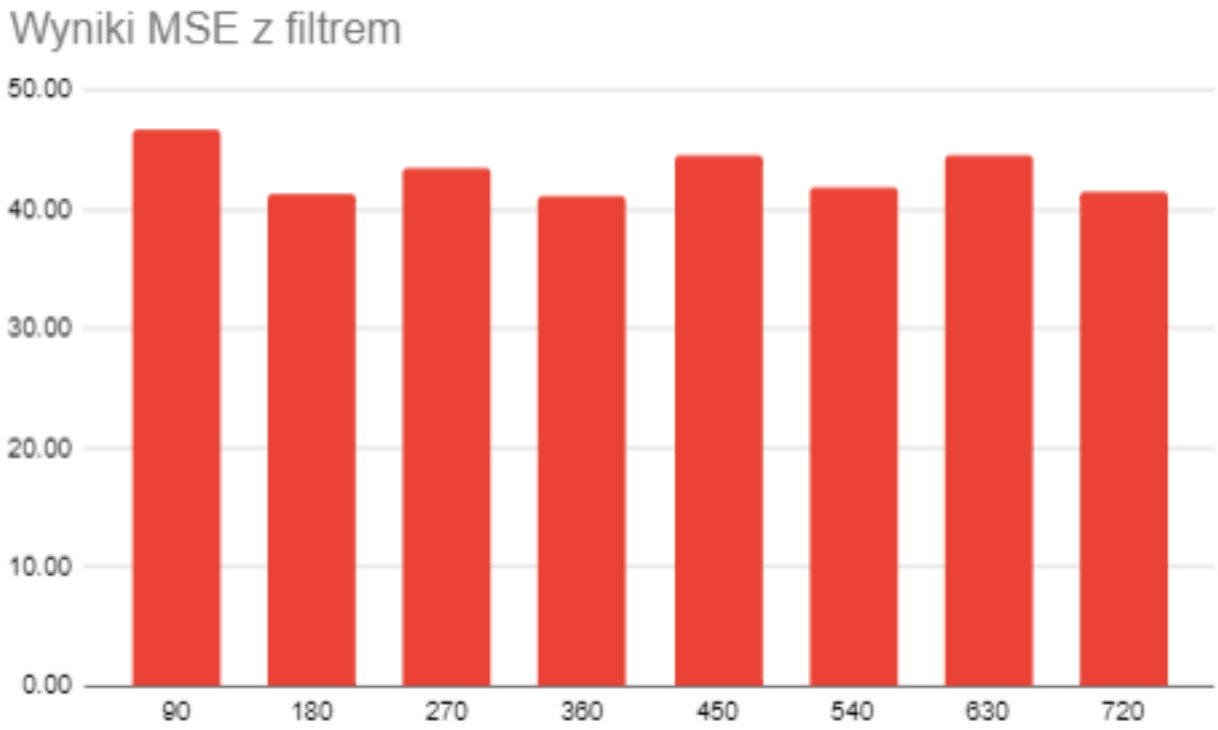
Wraz z każdą iteracją gdy rozpiętość wachlarza się zwiększa dochodzi do coraz mniejszego skupienia na elementach obrazu przez co po osiągnięciu pewnego progu obraz pogarsza się zyskując coraz większą ziarnistość.

Dla każdego wariantu proszę przedstawić wykres pokazujący zależność RMSE (oś Y) od aktualnej wartości zmienianego parametru (oś X) oraz krótko skomentować zaobserwowany przebieg

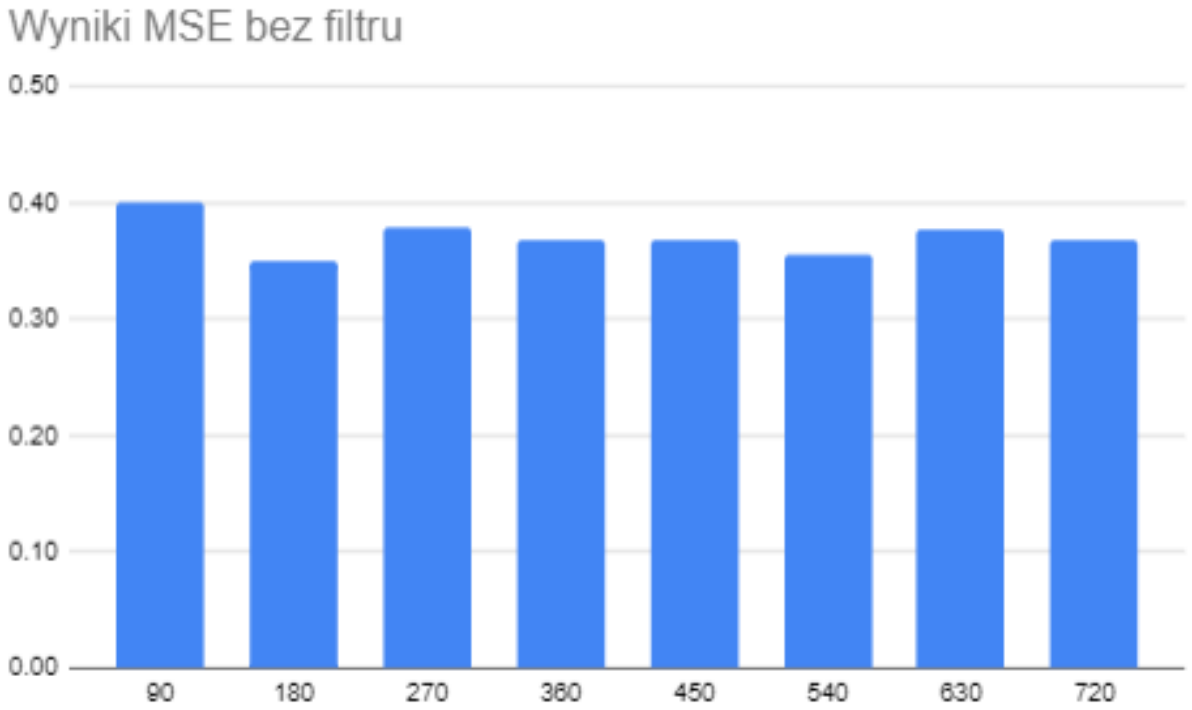
Zmiana liczby detektorów:

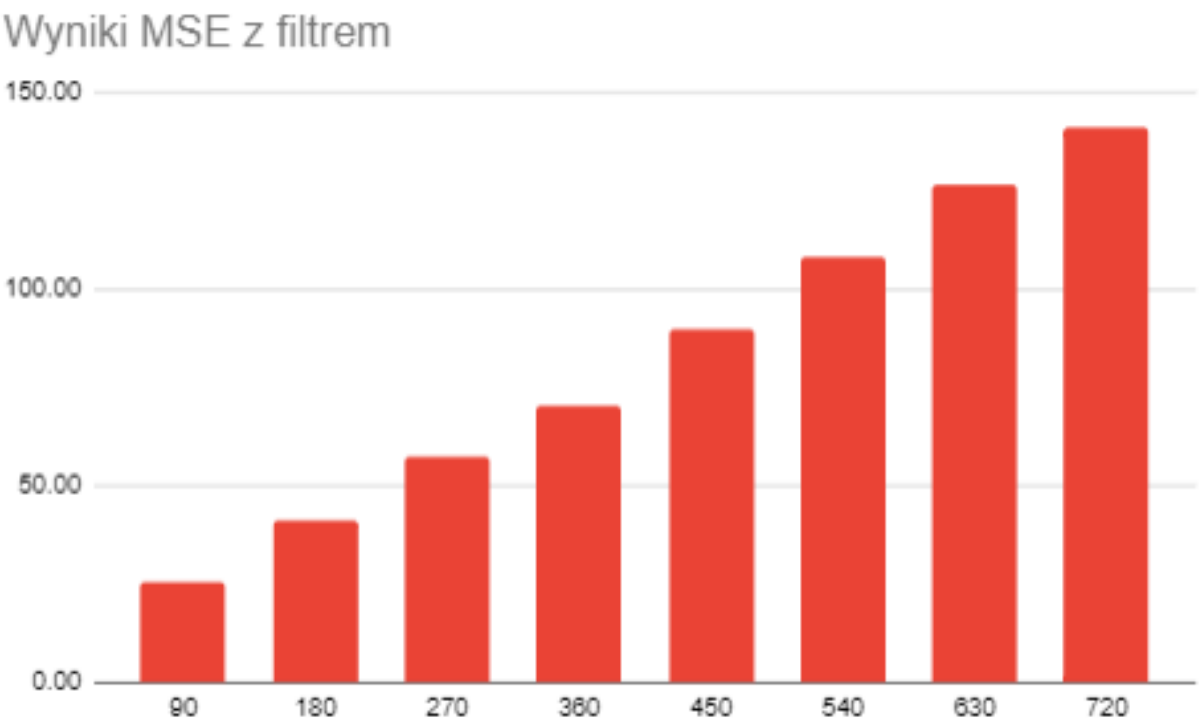


z

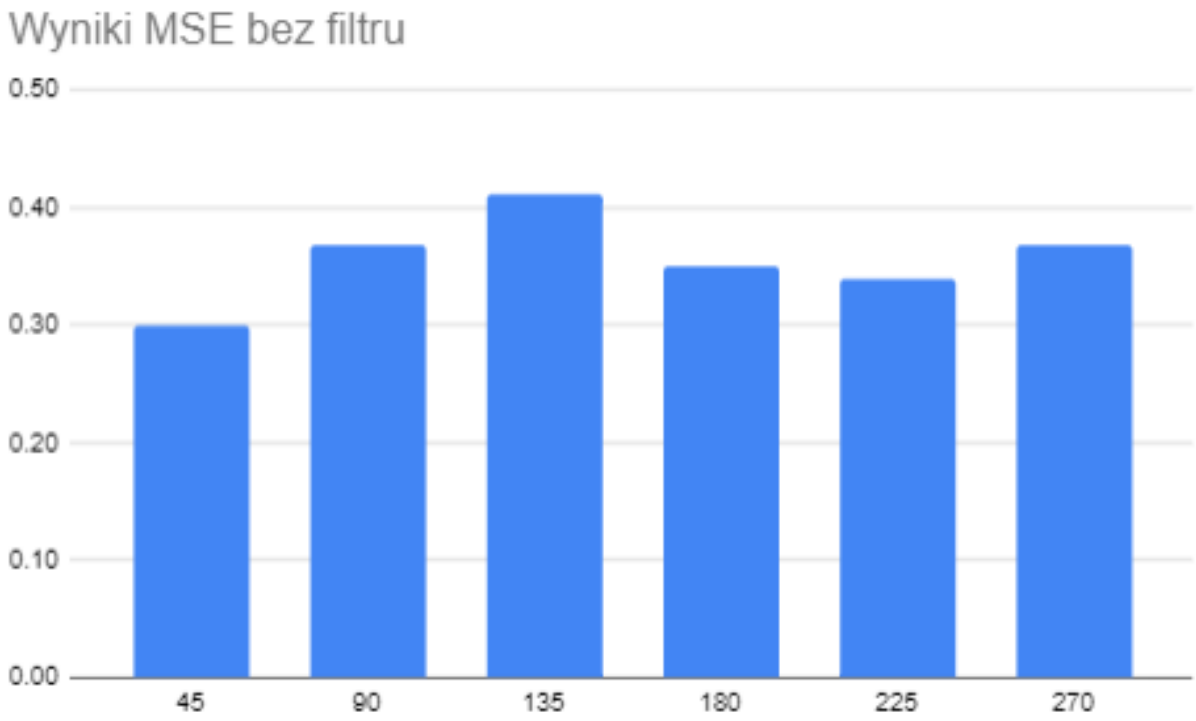


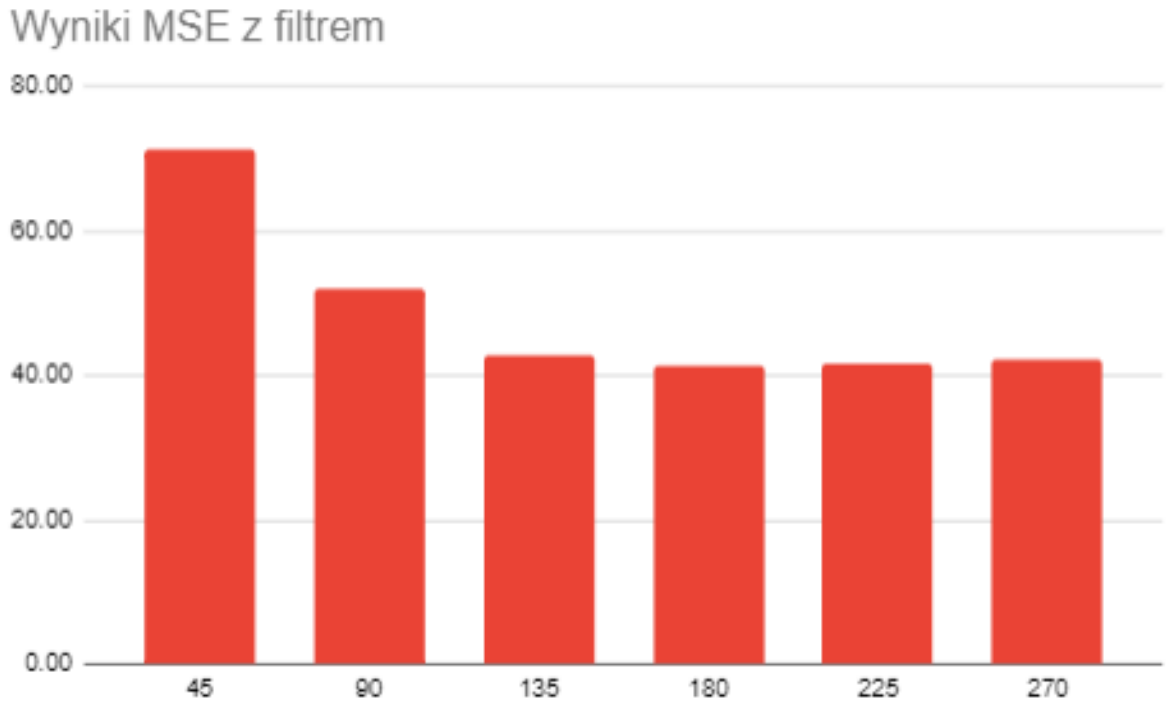
Zmiana liczby skanów:





Zmiana rozpiętości wachlarza:



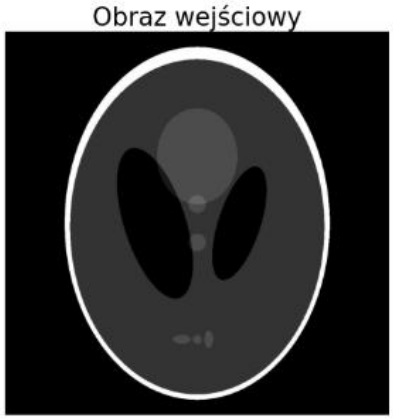


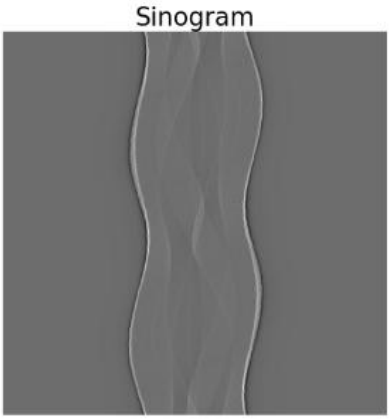
Wyniki RMSE bez filtra są we wszystkich 3 przebiegach podobne. Nie możemy zauważyć dużych wzrostów i spatków, jedynie niewiele mniejsze wartości na początku wykresów. RMSE nie oddaje jednak charaktreru uzyskanych obrazów. Inaczje sprawa ma się dla RMSE z

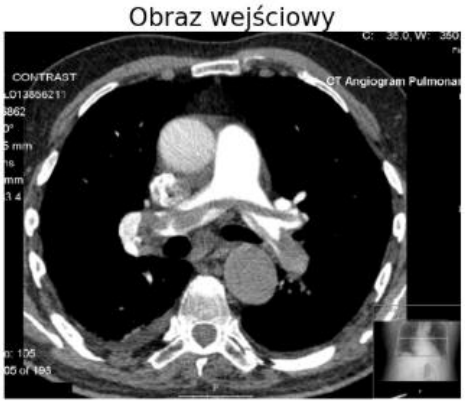
filtrem przy zmianie iteracji oraz rozpiętości wachlarza. Można zobaczyć korelacje z poprawa jakości przy większej liczbie iteracji, oraz pogorszeniu się wartości przy większym kącie rozpiętości wachlarza.

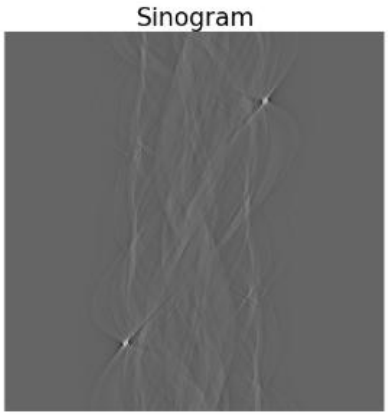
Dla dwóch wybranych obrazów oraz następujgcych parametrów: liczba detektorów = 360, liczba skanów = 360, rozpiętość wachlarza = 270 stopni, proszę wykonać dwa warianty obliczeń -- z włączonym i wyłączonym filtrowaniem sinogramu. Dla każdego obrazu proszę zaprezentować RMSE dla obrazu bez filtrowania i z filtrowaniem oraz krótko skomentować różnice w jakości między obrazami.

Obraz	*Liczba Detektorów*	*Liczba skanów*	*Rozpiętość wachlarza*	*Wynik MSE (bez filtra)*	*Wynik MSE (z filtrem)*
Sheep_logan	360	360	270	0.4125071579541798	69.49699266683835S
SADDLE_PE	360	360	270	0.3624215707850353	133.37242722687523?









Zastosowanie filtra ma znaczący wpływ na poprawę otrzymanego obrazu. Filtr pozwolił nam się pozbyć patternu z obrazu wynikowego. Dzięki temu gołym okiem widać poprawę jakości. Z kolei odnosząc się do błędu średniokwadratowego wyciągnęliśmy wniosek, że nie jest on satysfakcjonującym odzwierciedleniem jakości obrazów. Widać różnice między wynikami z filtrem i bez filtra ale nie są one aż tak duże jak wizualna różnica między otrzymanymi wynikami.