Homework sheet 1 2023-10-20

Due date: 2023-05-02 08:59

The goal of this homework, is to create your first CT reconstruction. You will get familiar working with sinograms, do common preprocessing tasks in CT, and finally implement your first reconstruction algorithm.

Delivery format

- Create a folder called "hw01" under the "homework" folder in your git repository.
- Answer all questions to the related to the homework in a file called "README.md" in the above created folder. In this file, you should document your findings, described paths you took and answer our questions.
- Put all scripts and images you've created to reproduce and comprehend your answers in the same folder. Examples for this, are scripts to create plots, visualization, certain reconstructions to compare different strengths and weaknesses and similar things.
- The concrete implementations of the homework, must be put into the "aomip" folder. You are free to choose the structure, layout and form of your implementation.
- Once you've finished your homework, commit your changes and create a git tag with `git tag -a "hw01" -m "Tag for homework 1"` and push your changes. This needs to be tagged and pushed to your "master" branch before the deadline ends.

Note

Generally, we expect and reward both explorativ work and good software engineering. Hence, we highly encourage you to look at the provided resources, do your own research and try different things and play around with the tools provided to you during the course.

Homework 1: Setup

If you haven't got everything setup properly yet, do it now! Check your project for instructions.

Homework 2: Literature

We encourage you to get the following books on CT reconstruction and imaging:

- Principles of Computerized Tomographic Imaging by Kak and Slaney (1988)
- The Mathematics of Computerized Tomography by Frank Natterer (1986)
- Computed Tomography: Algorithms, Insight, and Just Enough Theory, by Per Christian Hansen Jakob Sauer Jørgensen, and William R. B. Lionheart (2021)

You can use all of these books to read up on certain topics of CT reconstruction. You are not expected to read them cover to cover, especially not in the first week. But they will help you understand the problem further.

Homework 3: Preprocessing for computed tomography

Preprocessing is an integral part of the reconstruction process. Some measurements errors are very hard to fix during the reconstruction, but relatively easy in preprocessing. Here, we will also explore and simulate typical artifacts for CT reconstructions, which usually appear due to some errors in the measurements.

Note: The data from the Helsinki Tomography Challenge won't require immense amounts of preprocessing, as some steps are already performed, and hence we do not need to take care of it anymore. This includes flat field correction, center of rotation correction, and transmission to absorption conversion. However, some other preprocessing steps might still be helpful

You are encouraged to try some of the extra provided datasets and play around with the preprocessing steps with those datasets as well.

i) Flat-field correction

The real world sadly isn't as picture perfect as our mathematical world. One common problem during the scanning process is the *dark current* of the detector. Basically, this this means that a non-zero signal value is reported for a pixel, even though the X-ray source is turned off. A image taken without the X-ray source turned on is referred to *dark frame*.

Another effect of the detector, we'd like to compensate is a difference in response when the X-ray source is turned on. This can be captured using so called *flat-field images*, where no object is between the source and detector and hence a white or flat image is captured. In practice, this image might not be perfectly even, and hence can deteriorate the measurements.

The flat-field correction then simply is:

$$b = \frac{y - d}{f - d} \tag{1}$$

where y is the output image of the detector, d is the dark frame, and f is the flat field image. All operations here are coefficient wise.

Implement a function that takes the measurements, a dark frame, and a flat field image, and performs the above correction. If you can, find a dataset, on which this preprocessing makes a (noticeable) difference.

ii) Transmission to Absorption conversion

Simply speaking, what we measure is the transmission of the X-rays through the desired object. The measured intensity I_1 at a single pixel is given by the line integral the following way:

$$I_1 = I_0 \exp(-\int_0^s \mu(x)dx) \tag{2}$$

Here, I_0 is the initial intensity of the X-ray beam, μ is a function of the attenuation coefficients (which we seek to reconstruct!). One can rearrange the equation to:

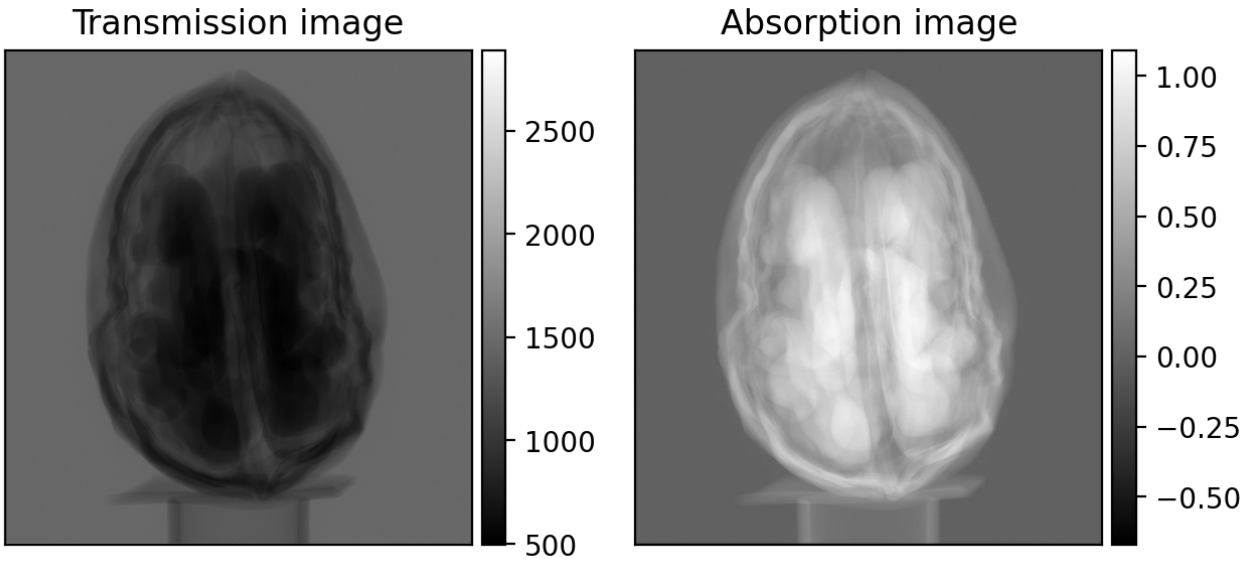

Figure 1: Comparison of a transmission image (left) and an absorption image (right)

$$-\log(\frac{I_1}{I_0}) = \int_L \mu(x)dx \tag{3}$$

This is known as the *negative log transform*. An example of a transmission image of a walnut, and it's corresponding negative log transform is given is Figure 1. In the absorption image, the higher the value, the more absorbent it is.

Implement the negative log transform, and the "inverse" operator, i.e. going from absorption to transmission data (which is useful for simulations).

The python function should look something like the following:

```
def transmission_to_absorption(x, I0):
    pass

def absorption_to_transmission(x, I0):
    pass
```

As I_0 is not necessarily know, you also need a way to estimate the initial intensity. In the ideal case, you can take some white pixel from the transmission image as I_0, however due to noise, this might not be a good value. Hence, take an empty region (i.e. without the object visible) of the transmission image, and average that to get an good estimate of the initial intensity.

You might want to write a function for that as well.

iii) Cleaning the signal

As you can see in Figure 1, the absorption image as negative values, which doesn't really make sense. What is a negative absorption? In this case, this is most likely due to a) noise, or b) some faulty in the detector. An easy fix to implement this, is to truncate either the transmission image, or the absorption image. In the transmission image everything above the initial intensity I_0 does not make sense, so it can be set to I_0. Or in the absorption image, everything below 0 can be set to zero.

Implement this truncation. How does it effect the processing? Can you find an example on how it affects the reconstruction (once you've finished Homework 4)?

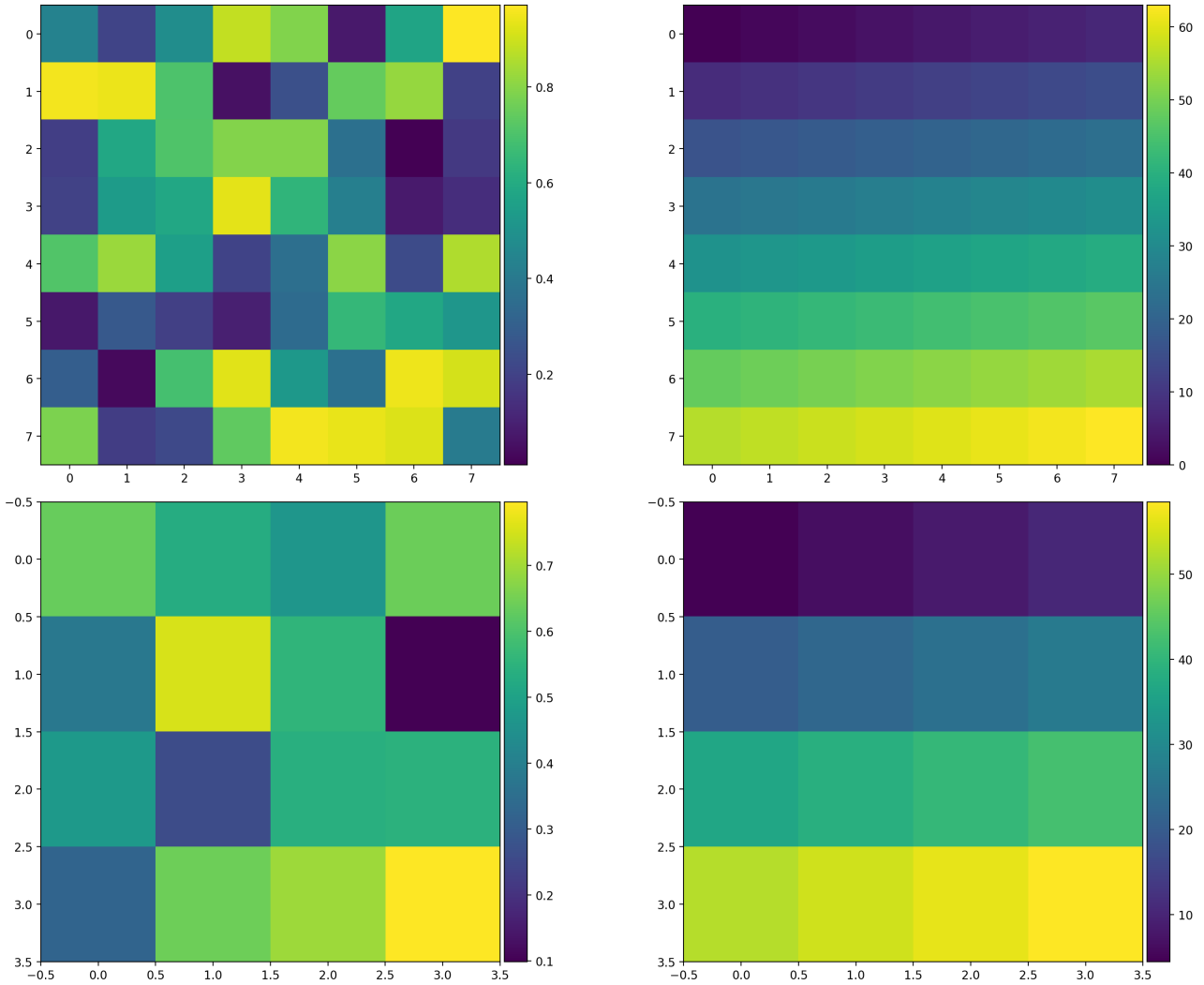

Figure 2: Two different 2D images (top) and their respective binned images (bottom). The binning factor was 2 in both cases.

iv) Binning

Binning is a useful tool for saving space and computation, especially for larger 3D datasets. Binning is the averaging of multiple pixels to one single pixels.

As an example, given a single: $\begin{bmatrix}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8\end{bmatrix}$ and a binning factor of 2, the resulting binned signal would be $\begin{bmatrix}1.5 & 3.5 & 5.5 & 7.5\end{bmatrix}$. The binning factor should always be a power of two.

In the 2D case, binning is the average of a square pixels. The size of the square is determined by the binning factor. And example for the 2D case is given in Figure 2.

Implement a function that performs binning on a 1D or 2D signal/image.

v) Center of rotation correction

Due to misalignments, the object to reconstruct may not be perfectly centered around the center of rotation. This can cause artifacts.

The fix is simple, move all projections a little to the side. NumPy provides the useful function `roll`, which you can use to implement a function that correct the center of rotation py rolling the projections by a given number of pixels on the last axis.

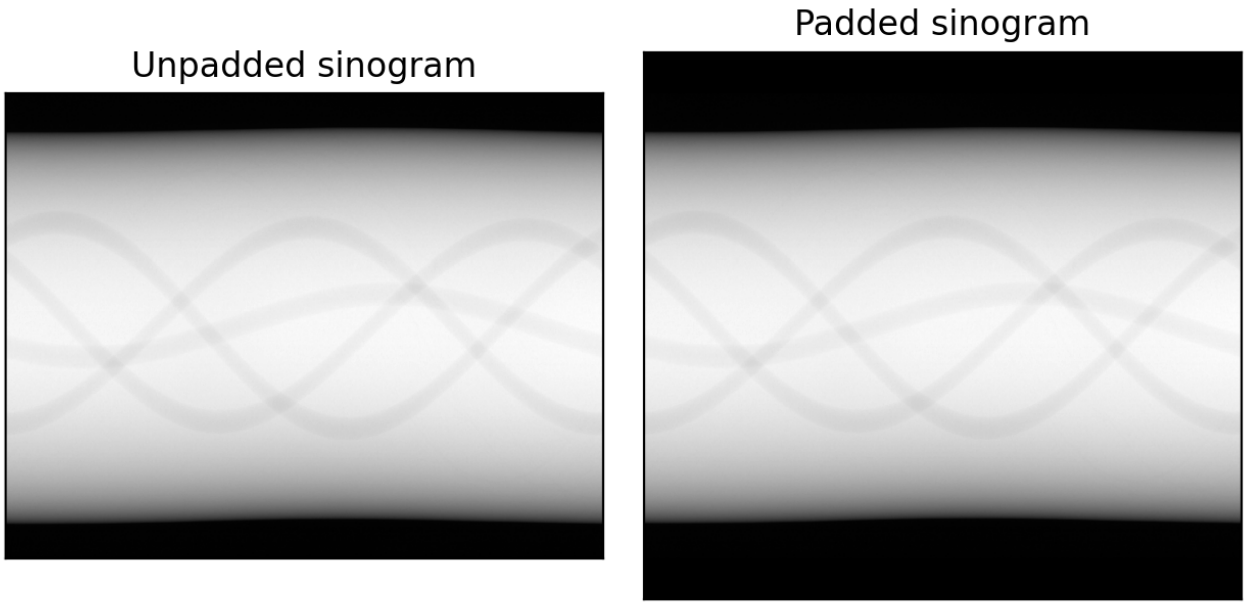

Figure 3: Adding zero padding to a sinogram. Left unpadded sinogram, right padded sinogram. The sinogram was padded by 100 pixels on each side

vi) Padding

Write a function that adds the given amount of zero-padding on all sides of a projection. This is similar virtually enlarging the detector. When would that be helpful? Can you find an example reconstruction, where this improves reconstruction quality? An example of padding is given in Figure 3.

Homework 4: Analytical Reconstruction

Now with the perfect preprocessed data available to us, we need to perform a reconstruction. In the tomographic reconstruction space, there is one widely used reconstruction technique called *filtered backprojection* (FBP). There exists a mathematical derivation of it, however, we will not focus on that here. The interested student can read on that in the reference given above.

As you have seen in the exercise in the lesson, the backprojection is a blurry version of the original object. Hence, it is a logical step to perform a filtering of either the sinogram and then do the backprojection, or perform a filtering of the backprojection. The former method is know as FBP, the latter one backprojection then filter.

Here, we will focus on the first method. The FBP is the backprojection of the convolution of the sinogram with a filter:

$$\hat{x} = A^* \left(b * h \right) \tag{4}$$

where $*$ is the convolution of two signals, and A^* is the backprojection operator. One of the most efficient implementations of the convolution is the coefficient wise multiplication in the Fourier space. Hence, the above can be rewritten as:

$$\hat{x} = A^* \, F^{-1} \, H \, F \, b \tag{5}$$

where, F is the Fourier Transform and H is the filter in the fourier domain.

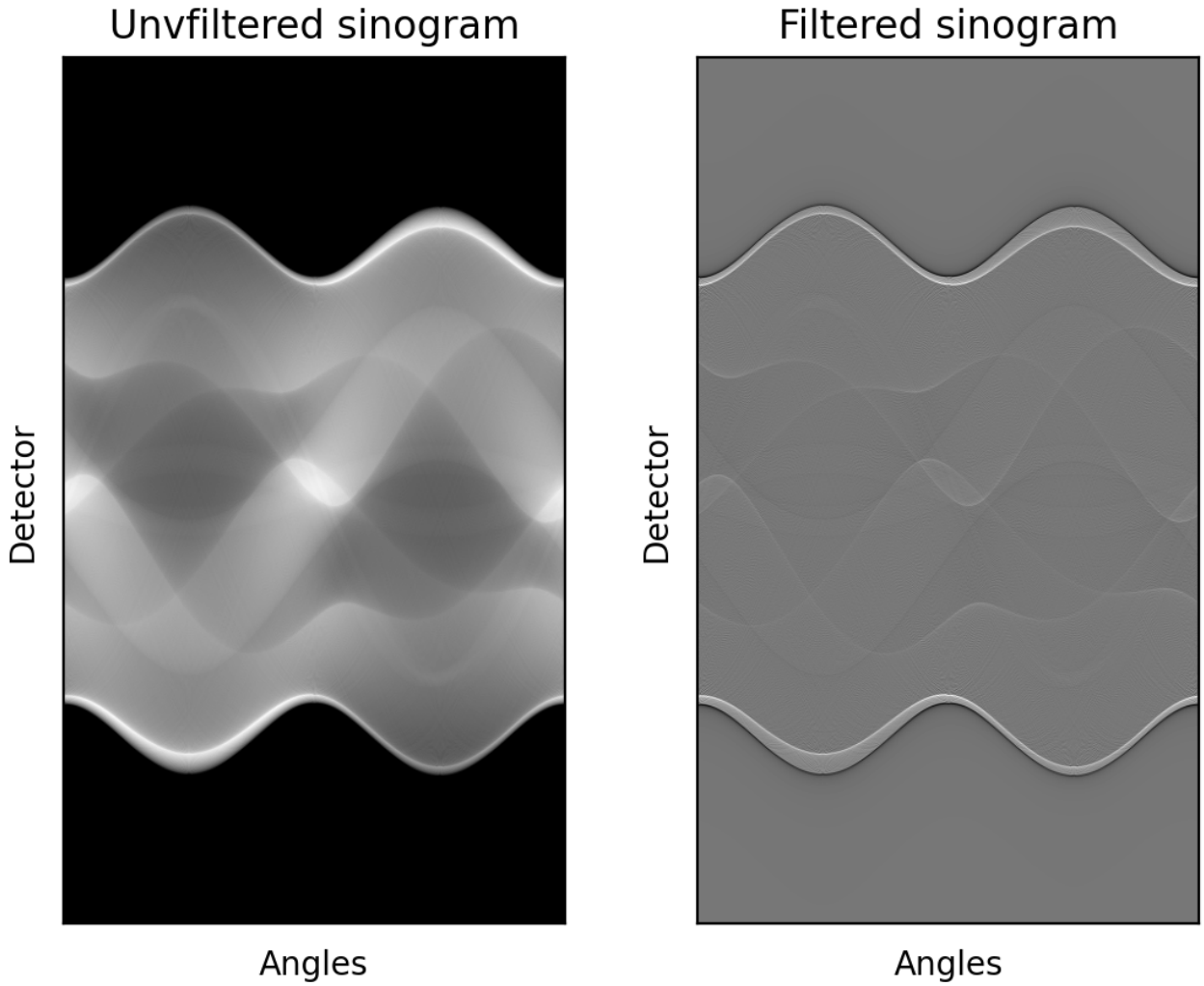

Figure 4: Sinogram of the Shepp Logan phantom (360° arc, with 420 positions) on the left, and the filtered sinogram (ideal ramp filter) on the right.

The rough steps for the FBP are (given a sinogram and some filter in the frequency domain): For each projection, transform it into the frequency domain using the *Fast Fourier Transform*, multiply it coefficient wise with the filter, then transform the result back using the *Inverse Fast Fourier Transform*. An example for filtered sinogram compared to the unfiltered one is give Figure 5. Backprojecting each of the sinograms, the results in Figure **??** are achieved.

The first filter is the ideal ramp filter, or Ram-Lak filter (this is mostly used in the context of CT). The ramp filter is a high pass filter, with a linear response. Other filters exists, see Figure fig:filters for a visualization, and the references below for more filters.

Your assignment is to implement a method similar to MATLAB's `iradon` or scikit-images ones (references down below). You must implement the so called *Ram-Lak* filter, which is the ideal filter. Further, look at the filters provided by MATLAB or scikit-image and implement two other filters.

A note on the implementation: First, you should use NumPy's implementation of the FFT. Second, you might need to add padding to the sinogram to apply the FFT. It at least needs to be even, but padding to the next power of two can improve accuracy (experiment with it!). Further, much of the literature on the filters discussed the construction in the spatial domain. However, that is mostly the case, as it was more efficient back in the day. Nowadays, you can directly construct them in the

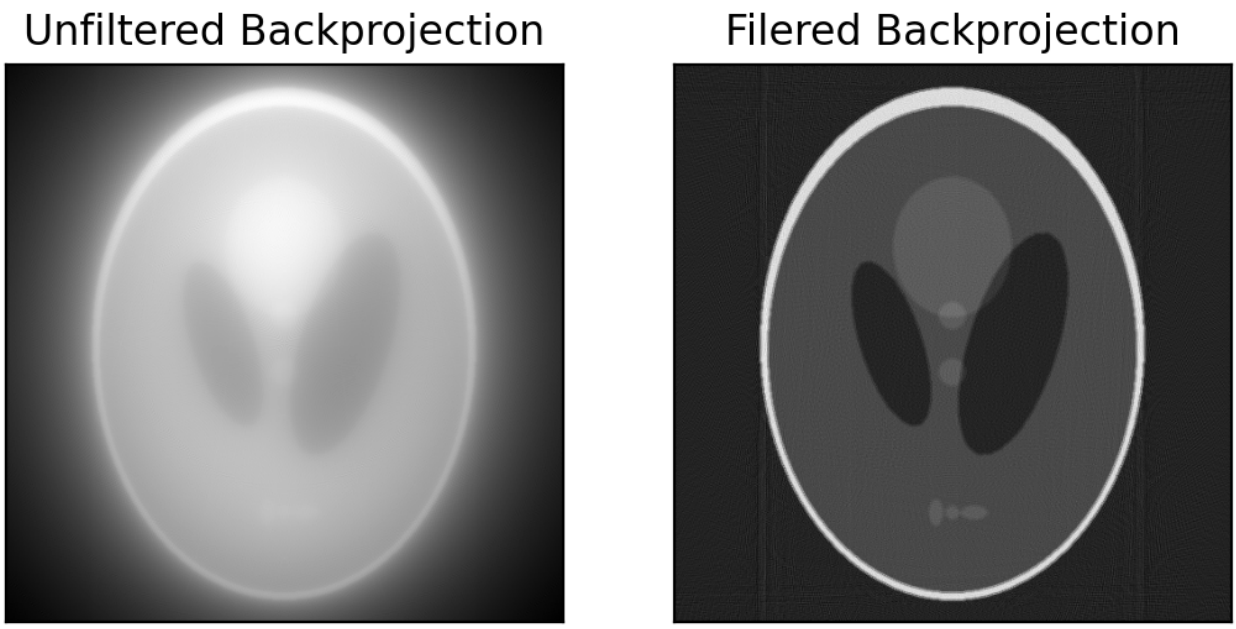

Figure 5: Unfiltered backprojection (left) and filtered backprojection (right) of the sinograms in Figure 5.

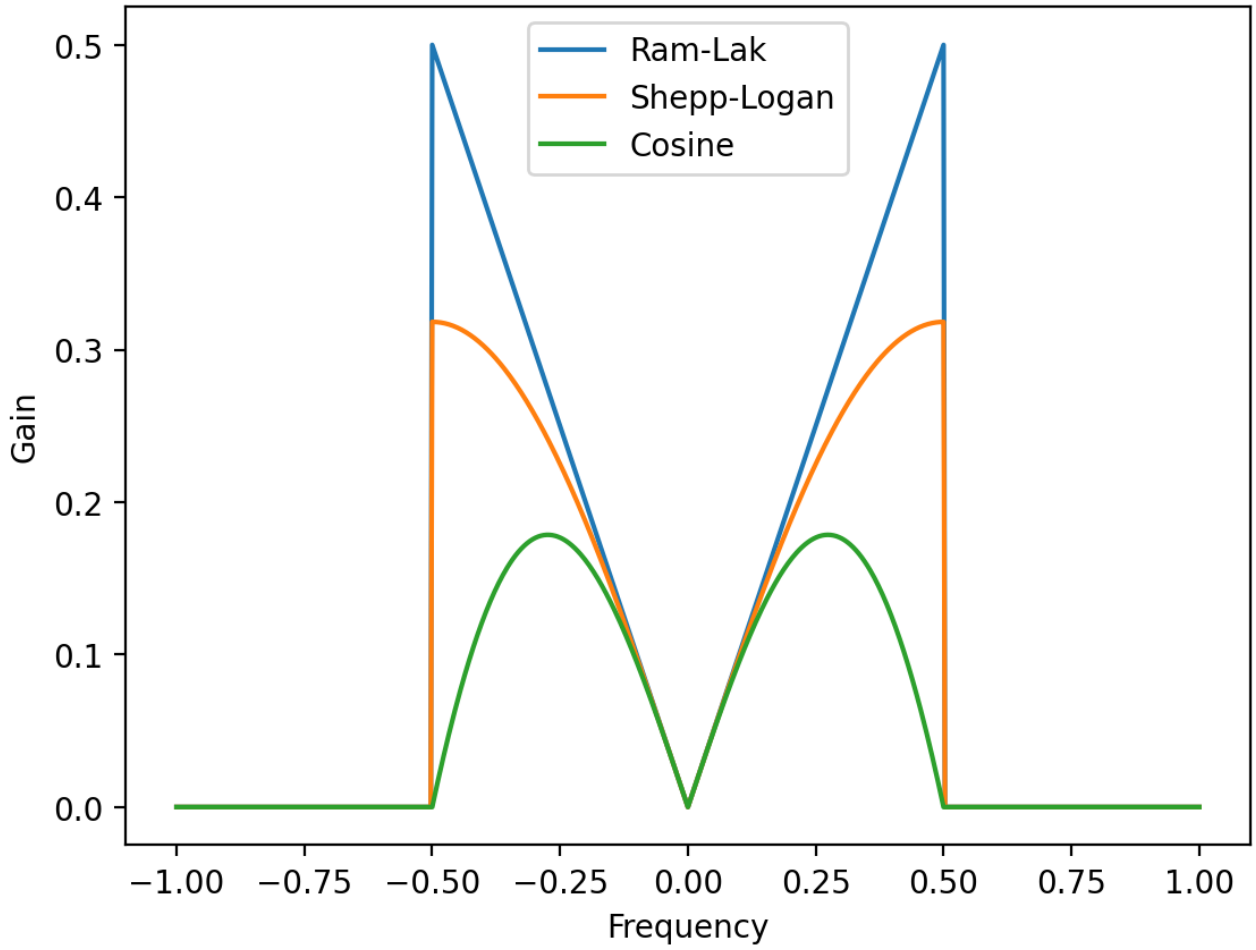

Figure 6: Visualization of 3 common filters used for the FBP

frequency space.

Optional: Cosine weights

From a mathematical standpoint, the FBP is only valid for parallel beam geometry. However, we usually deal with fan or cone beam data in this course. The FBP can be extended to accommodate for fan or cone beam data. The data needs to be multiplied with cosine weights first.

The weights can be computed with the equation for 1D and 2D projections respectively

$$\frac{D}{\sqrt{D^2+u^2}} \qquad \frac{D}{\sqrt{D^2+u^2+v^2}}$$

Here, D is the distance from the source to the center of the projector. u and v can be computed based on the detector coordinate $\begin{bmatrix} i & j \end{bmatrix}^T$:

$$u = (i+0.5)s_u - \frac{w}{2} \qquad v = (j+0.5)s_v - \frac{h}{2}$$

where s is the spacing of a detector pixel in u and v direction, w is the width of the detector and h is the height of the detector.

Now, you can multiply the projections with the cosine weights in the spatial domain. This step needs to be performed before you do the filtering. Check chapter 3.4 in Kak et. al for more detail.

References:

- https://www.mathworks.com/help/images/ref/iradon.html
- https://scikit-image.org/docs/stable/api/skimage.transform.html#skimage.transform.iradon

(Optional) Homework 5: Visualization

If you explore multiple datasets you will quickly realize, that it's useful to visualize sinograms, geometry, reconstructions, look at cross sections, scrub through 3D stacks and volumes. Proper visualization will help you find errors quicker and show results clearer.

This exercise encourages you to find ways to visualize different aspects of your workflow.

The best visualizations have the chance to be integrated in our open source tool!