Exercise Sheet 2 - Parallel-Beam

Andreas Maier, Jennifer Maier, Bastian Bier,
Alexander Preuhs, Christopher Syben

November 7, 2017

In this exercise, we will implement a parallel-beam filtered back-projection (FBP) algorithm and test it on the phantom, which we created in Exercise 1.

1. **Sinogram Generation:** Implement the Radon transform to create a parallel-beam sinogram of your phantom. Projections should be generated using the ray-driven approach for a fixed 180° with variable angular increment.

   The following input parameters are mandatory: number of projections, detector spacing, number of detector pixels and the angular range. You can assume that your source-to-detector distance is large enough to cover the full phantom. To read out your phantom at arbitrary positions, you need 2D interpolation (see the `InterpolationOperators` class).

2. **Backprojection**: Implement a pixel-driven back-projector. For each rotation angle, the back-projector needs to project each pixel position onto the detector, read-out the value and add it to the corresponding pixel. Use `InterpolationOperators` to compute a bilinear interpolation on the detector. How does the backprojection result look like? Explain this effect.

   See “Implementation Details” for more information.

3. **Ramp and RamLak filter**: Implement a row-wise ramp and a RamLak filter on a Grid2D using Fourier transforms. Start implementing the ramp filter in Fourier domain. Then, implement the RamLak filter in spatial domain. Compare both implementations.

   Use the implemented `Grid1DComplex` class and its `transformForward()` and `transformInverse()` methods.

   See “Implementation Details” for more information.

4. **Filtered Back Projection**: Combine the back-projector and the filter to create a reconstruction. For this task, we need to filter the projection data and then use back-projection. The filtering will be implemented in the frequency domain as a multiplication operation of the 1D Fourier transform of the projection data and a ramp kernel. (How does the ramp filter look

like in the frequency/spatial domain?). Two functions are required for our FBP implementation, which are finally combined.

Think you are done? Checklist:

- ☐ Sinogram created (have you varied the parameters?)
- ☐ Backprojected the sinogram (observations?)
- ☐ Implemented and applied ramp filter (artifact?)
- ☐ Implemented and applied RamLak filter (artifact gone?)
- ☐ Validated by a supervisor

Implementation Details

Pixel-Driven Back-Projection The back-projection should be able to deal with the sinograms you generated in Exercise 1. That means you should incorporate, e.g., the detector spacing, and other variables from your sinogram. It also takes the size and pixel spacing of the reconstructed image as inputs. The origin can be assumed to be in the center of the image.

Ramp Filter Ramp filtering is a convolution of each detector line of your sinogram with the ramp-filtering kernel. Because the ramp filtering kernel is best known in Fourier domain we perform the convolution by element-wise multiplication in the Fourier domain. Some details are important to define the ramp-filtering kernel:

1. FFT algorithms swap the positive and negative frequency axes. The vector you obtain by the forward FFT starts with the zero frequency up to the positive maximum located in the center of the vector, then it continues from the negative maximum to almost zero at the end of the vector. (Hint: That means if you visualize your kernel using the `show()` method, it should look like a pyramid.)

2. For proper ramp-filtering we need to apply zero padding. Casting a `Grid1D` to a `Grid1DComplex` applies zero-padding automatically, e.g., if you have a detector length of 400, the class first rounds up to the nearest power of two and then doubles the length. That means you would get 1024 complex values after Fourier transform (have a look at the constructor!). Bear in mind that the ramp filter needs to be defined over the full length in Fourier domain, i.e., the 1024 values. The same holds for the Ram-Lak filter except that it is implemented in spatial domain and then Fourier-transformed.

3. To compute the ramp kernel you need to know the spacing of your frequency axis. This depends on the amount of zero-padding and your detector spacing. It can be computed by:

$$\Delta f = \frac{1}{\Delta s \cdot K} \ , \tag{1}$$

   where Δf is the frequency spacing, Δs is the detector spacing and K is the length of your signal after zero-padding.

4. The `setAtIndex()` method of Grid1DComplex is not aware of the complex nature of the grid. Use `setRealAtIndex()` and `setImagAtIndex()` instead.

5. Recall complex multiplication!

6. RamLak filters are initialized in spatial domain. Use the formula from the lecture to initialize the filter in a Grid1DComplex.