Sparsity-based algorithms for inverse problems

Ganguly, P.S.

Chapter 2

Implementation-adapted filters for synchrotron tomography

2.1 Introduction

In several scientific disciplines, such as materials science, biomedicine and engineering, a quantitative three-dimensional representation of a sample of interest is crucial for characterizing and understanding the underlying system [24]–[27]. Such a representation can be obtained with the experimental technique of computerized tomography (CT). In this approach, a penetrating beam, such as X-rays, is used to obtain projection images of a sample at various angles. These projections are then combined by using a computational algorithm to give a 3D reconstruction [28], [29].

Different tomographic setups are used in various practical settings. Our focus here is on tomography performed with a *parallel-beam* X-ray source at synchrotrons. Synchrotrons provide a powerful source of X-rays for imaging, enabling a broad range of high-resolution and high-speed tomographic imaging techniques [30]–[32].

A typical tomography experiment at the synchrotron can be described by a pipeline consisting of several sequential steps (see Fig. 2.1). First, a sample is prepared according to the experiment and imaging setup requirements. Then, the imaging system is aligned [33], and a series of projection images of the sample are acquired [34]. These data are then processed for calibration, contrast improvement (e.g. phase retrieval [35]) or removal of undesirable artefacts like rings or stripes [36]. Following pre-processing, the data are fed into a reconstruction software package that makes use of one or more standard algorithms to compute a 3D reconstruction [37], [38]. The reconstruction volumes can then be further post-processed and analysed [39], [40] to obtain parameter estimates of the system being

This chapter is based on:

Improving reproducibility in synchrotron tomography using implementation-adapted filters. *P. S. Ganguly, D. M. Pelt, D. Gürsoy, F. de Carlo, and K. J. Batenburg*. Journal of Synchrotron Radiation 28, no. 5, 2021.

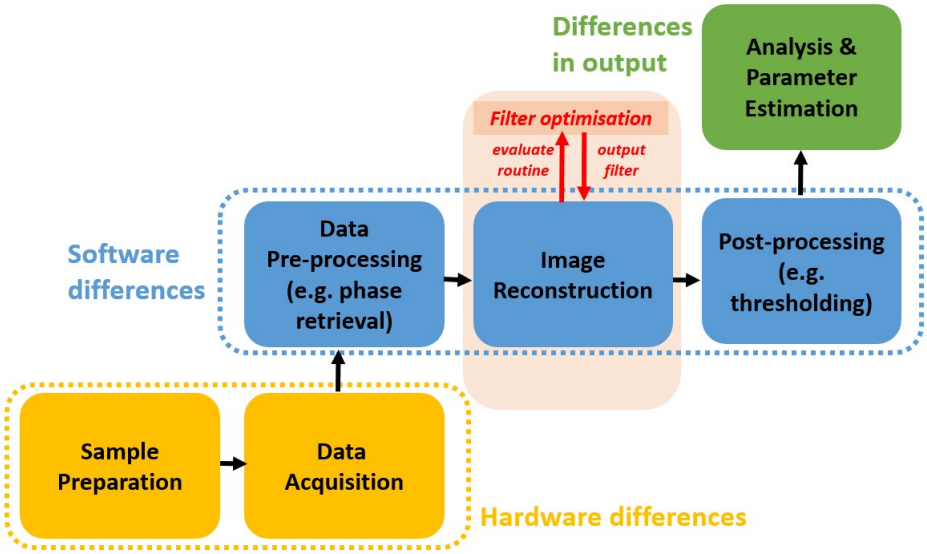

Figure 2.1: Schematic representation of a typical tomography pipeline at synchrotrons. Hardware differences play an important role during sample preparation and data acquisition. Software differences affect image pre-processing, reconstruction and post-processing. Together these lead to differences in the output of analysis and parameter estimation studies. In this chapter we propose a filter optimization method that works as a wrap-around routine on the reconstruction block. Our method only requires evaluations of the reconstruction routine and does not require any internal coding. The output of our method is a filter that can be used in the reconstruction block for more reproducible reconstructions.

studied. In some cases, systematic imperfections in the data can also be corrected by post-processing reconstructions. For example, ring artefacts, which are commonly observed in synchrotron data, can be corrected before or after reconstruction [37].

At various synchrotron facilities in the world, the pipeline described above is implemented using different instruments, protocols and methods specific for each facility [41]. These differences are on the level of both hardware and software. Dissimilarities in the characteristics of the used X-ray source and detection system, including camera, visible light objective and scintillator screen, lead to differences in the acquired data. The differences in the data are then further compounded by variations in processing and reconstruction software, resulting in differences in voxel or pixel intensities, and eventually in variations in the output of post-processing and analysis routines.

For users, such differences pose several challenges. First, it is difficult to ensure that results and conclusions obtained from experiments at one facility are comparable and consistent with experiments from another facility. Second, other researchers seeking to reproduce the results of a previous work with their own software might not be able to do so, even if they have access to raw data. In [41], the authors report quantitative

differences at various stages of the pipeline when scanning the same object at different synchrotrons. Reproducibility and the ability to verify experimental findings is crucial for ascertaining the reliability of scientific results. Therefore, in order to ensure reproducibility for the synchrotron pipeline, it is important to quantify and mitigate differences in the acquired, processed and reconstructed data.

Hardware and software vary across synchrotrons for a number of reasons. Each synchrotron uses a pipeline that is optimized for its specific characteristics. In addition, legacy considerations play a role in the choice of components. Because of the variations across synchrotrons, any successful strategy for creating reproducible results must take this diversity into account. Ideally, the choices for specific implementations of each block in the synchrotron pipeline in Fig 2.1 should not influence the final results of a tomography experiment. Following this strategy, each block can be optimized for reproducibility independently from the rest of the pipeline.

In this chapter, we focus on improving the reproducibility of the reconstruction block in the pipeline. In most synchrotrons, fast analytical methods such as filtered backprojection (FBP) [29] and Gridrec [42] are the most commonly-used algorithms for reconstruction. This is primarily because such algorithms are fast and work out-of-the-box without parameter tuning. These algorithms give accurate reconstructions when the projection data are well-sampled, such as in microCT beamlines where thousands of projections can be acquired in a relatively short time.

Several open-source software packages for synchrotron tomography reconstruction are available, such as TomoPy, the ASTRA toolbox and scikit-image [37], [43], [44]. Usually, an in-house implementation of FBP or Gridrec, or one of the open-source software packages is used for reconstruction. Each of these implementations contains a *filtering* step that is applied to the projection data as part of the reconstruction. Filtering influences characteristics, such as noise and smoothness, of the reconstructed volume. A sample-independent, pre-defined filter is generally used for reconstruction. Some filters used in this step have tunable parameters, but these are often tuned on-the-fly and are not recorded in metadata.

Reconstructions in analytical algorithms are obtained by inversion of the Radon transform [45]. Although this inversion is well-defined mathematically in a continuous setting, software implementations invariably have to work in a discretized space. In software implementations, the measurements as well as the reconstructed volume are *discrete*. In a discretized space, inversion of the Radon transform often translates to a *backprojection* step, which makes use of a discretized *projection kernel* to simulate the intersection between the scanned object and X rays [46]. The backprojection operation can also be performed directly using interpolations in Fourier space [29].

Different choices of discretization and interpolation, in projection kernels and filters, are possible. These choices lead to quantitative differences between the reconstructions obtained from different software implementations. A simple example of this effect is shown in Fig. 2.2, where we consider a phantom of pixel size 33×33 and data along 8 projection angles uniformly sampled in $[0, \pi)$. We compare reconstructions of the same data using two different projection kernels and two different filtering methods. In both instances, the image to be reconstructed contains a single bright pixel at the centre of the field-of-view. The *sinogram* of such an image (i.e. the combined projection data for the

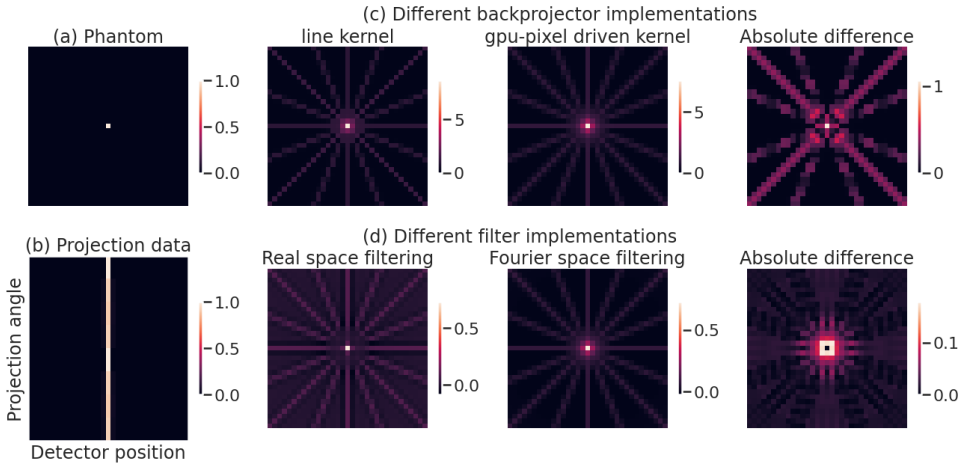

Figure 2.2: Differences in reconstruction due to differences in backprojector and filter implementations. (a) a 33×33 phantom with one bright pixel, (b) sinogram of the phantom (computed using a strip kernel from the ASTRA toolbox), (c) differences in (unfiltered) backprojection when using different backprojectors: (*left to right*) backprojection using a CPU line kernel from the ASTRA toolbox, backprojection using a GPU pixel-driven kernel from the ASTRA toolbox, absolute difference between the two backprojections. (d) differences in reconstruction when using different filtering routines in FBP with the `gpu-pixel` kernel as backprojector: (*left to right*) reconstruction using filtering in real space with the Ram-Lak filter, reconstruction using the ramp filter in Fourier space, absolute difference between the two reconstructions.

full range of angles) was computed using a CPU strip kernel projector from the ASTRA toolbox [43]. Backprojections of this projection data using two other projectors - a CPU line projection kernel and a pixel-driven kernel implemented on a graphics processing unit (GPU) - show significant, radially-symmetric differences. These differences are dependent on the number of projection angles used, and are highly structured, unlike differences due to random noise. We also observe structured differences between reconstructions when the same projection kernel (`gpu-pixel`) is used after different filtering operations in real and Fourier space. This example highlights the impact of discretization and interpolation choices on the final reconstruction obtained from identical raw data.

Our main contribution in this chapter is a heuristic approach that can improve reproducibility in reconstructions. Our method consists of optimizing the filter used in different software implementations of reconstruction methods. We call such optimized filters *implementation-adapted filters*. The computation of our filters does not require knowledge of the underlying software implementation of the reconstruction algorithm. Instead, a wrapper routine around any black-box implementation can be used for filter computation. Once computed, these filters can be applied with the reconstruction software like any other standard filter.

Our chapter is organized as follows. In Section 5.2, we formulate the reconstruction

problem mathematically and discuss the effect of different software implementations. In Section 2.3, we describe our algorithm for computing implementation-adapted filters. Numerical experiments described in Sections 2.4 and 4.5 demonstrate use cases for our filters on simulated and real data. Finally, we discuss extensions to the current work in Section 5.5 and conclude our chapter in Section 4.6. Our open-source Python code for computing implementation-adapted filters is available on GitHub (https://github.com/poulamisganguly/impl-adapted-filters).

2.2 Background

2.2.1 Continuous reconstruction

Consider an object described by a two-dimensional attenuation function $f : \mathbb{R}^2 \to \mathbb{R}$. Mathematically, the tomographic projections of the object can be modelled by the Radon transform, $\mathcal{R}(f)$. The Radon transform is the line integral of f along parametrized lines $l_{\theta,t} = \{(x, y) \in \mathbb{R}^2 \,|\, x\cos\theta + y\sin\theta = t\}$, where θ is the projection angle and t is the distance along the detector. Projection data $p_\theta(t)$ along an angle θ are thus given by

$$p_\theta(t) = \mathcal{R}(f) = \iint_{\mathbb{R}^2} f(x, y)\delta(x\cos\theta + y\sin\theta - t)dxdy. \tag{2.1}$$

The goal of tomographic reconstruction is to obtain the function $f(x, y)$ given the projections $p_\theta(t)$ for various angles $\theta \in \Theta$. One way to achieve this is by direct inversion of the Radon transform. Given a complete angular sampling in $[0, \pi)$, the Radon transform can be inverted giving the following relation [29]

$$f(x, y) = \int_0^\pi \left(\int_{-\infty}^{\infty} \tilde{P}_\theta(\omega)|\omega| e^{2\pi i\omega(x\cos\theta + y\sin\theta)} d\omega \right) d\theta, \tag{2.2}$$

where $\tilde{P}_\theta(\omega)$ denotes the Fourier transform of the projection data $p_\theta(t)$ and multiplication by the absolute value of the frequency $|\omega|$ denotes filtering with the so-called ramp filter.

For noiseless and complete data, the Radon inversion formula (2.2) provides a perfect analytical reconstruction of the function $f(x, y)$ from its projections. However, in practice, tomographic projections are obtained on a *discretized* detector, consisting of individual pixels, and for a finite set of projection angles. Additionally, the reconstruction volume must be discretized in order to represent it on a computer. Therefore, in practical applications, a discretized version of (2.2) is used to obtain reconstructions.

2.2.2 Discrete reconstruction

Discretization of the reconstruction problem yields the following equation for the discrete reconstruction $r(x_d, y_d)$:

$$r(x_d, y_d) = \sum_{\theta_d \in \Theta} \sum_{t_d \in T} h(t_d) P_{\theta_d}(x_d\cos\theta_d + y_d\sin\theta_d - t_d), \tag{2.3}$$

where (x_d, y_d), θ_d and t_d denote discretized reconstruction pixels, angles and detector positions, respectively, and $h(t_d)$ is a discrete real-space filter. This inversion formula is known as the filtered backprojection (FBP) algorithm.

The FBP equation (2.3) can be written algebraically as the composition of two matrix operations: filtering and backprojection. Filtering denotes convolution in real space (or, correspondingly, multiplication in Fourier space) with a discrete filter. Backprojection consists of a series of interpolation and numerical integration steps to sum contributions from different projection angles. These discretized operations can be implemented in a number of different ways and different software implementations often make use of different choices for discretization and interpolation. Consequently, the reconstruction obtained from a particular implementation is dependent on these choices. The reconstruction $\boldsymbol{r}_I$ from an implementation I can thus be written as

$$\boldsymbol{r}_I(\boldsymbol{h}, \boldsymbol{p}) = \boldsymbol{W}_I^T \boldsymbol{M}_I(\boldsymbol{h}, \boldsymbol{p}), \tag{2.4}$$

where $\boldsymbol{W}_I^T$ is the backprojector and $\boldsymbol{M}_I(\cdot, \cdot)$ is the (linear) filtering operation associated with implementation I. We denote the discrete filter by $\boldsymbol{h}$.

In the following subsection, we discuss some common choices for projection and filtering operators in software implementations of analytical algorithms.

2.2.3 Differences in projectors and filtering

In order to discretize the Radon transform, we must choose a suitable discretization of the reconstruction volume, a discretization of the incoming ray and an appropriate numerical integration scheme. All these choices contribute to differences in different backprojectors $\boldsymbol{W}_I^T$ in (2.4).

Voxels (or pixels in 2D) in the reconstruction volume can be considered either to have a finite size or to be spikes of infinitesimal size. Similarly, a ray can be discretized to have finite width (i.e. a strip) or have zero width (i.e. a line). The numerical integration scheme chosen might be piecewise constant, piecewise linear or continuous. All of these different choices have given rise to different software implementations of backprojectors [46]. There exist different categorizations of backprojectors in the literature; for example, the `linear` kernel in the ASTRA toolbox is referred to as the slice-interpolated scheme in [47] and the `strip` kernel is referred to as the box-beam integrated scheme in the same work. In this chapter, we designate different backprojectors with the terms used in the software package where they have been implemented.

In addition to the choices mentioned above, backprojectors have also been optimized for the processing units on which they are used. For this reason, backprojectors that are optimized to be implemented on graphics processing units (GPUs) might be different from those that are implemented on a CPU due to speed considerations. In particular, GPUs provide hardware interpolation that is extremely fast, but can also be of limited accuracy compared to standard floating point operations.

So far, we have discussed real space backprojectors. Fourier-domain algorithms such as Gridrec [42] use backprojectors that operate in the Fourier domain. These operators are generally faster than real-space operators, and are therefore particularly suited for accelerating iterative algorithms [48]. Unlike real space backprojectors, Fourier-space

backprojectors perform interpolation in the Fourier domain. As this might lead to non-local errors in the reconstruction, an additional filtering step is performed to improve the accuracy of the interpolation.

Apart from differences in backprojectors, different implementations also vary in the way they perform the filtering operation in analytical algorithms. Filtering can be performed as a convolution in real space or as a multiplication in Fourier space. Real space filtering implementations can differ from each other in computational conventions, for example by the type of padding used [13] to extend the signal at the boundary of the detector. Moreover, the zero-frequency filter component is treated in different ways between implementations. For example, the Gridrec implementation in TomoPy sets the zero-frequency component of the filter to zero.

2.3 Implementation-adapted filters

We now present the main contribution of our chapter. In order to mitigate the differences between implementations discussed in the previous section, we propose to specifically tune the filter $\boldsymbol{h}$ for each implemented analytical algorithm. In the following, we describe an optimization scheme for the filter, which helps us to reduce the differences between reconstructions from various implementations.

We optimize the filter by minimizing the ℓ^2 difference with respect to the projection data $\boldsymbol{p}$. This can be stated as the following optimization problem over filters $\boldsymbol{h}$:

$$\boldsymbol{h}_I^* = \arg\min_{\boldsymbol{h}} \|\boldsymbol{p} - \boldsymbol{W}\boldsymbol{r}_I(\boldsymbol{h}, \boldsymbol{p})\|_2^2, \tag{2.5}$$

where $\boldsymbol{r}_I$ is the reconstruction from implementation I. Note that the forward projector $\boldsymbol{W}$ used above is chosen as a fixed operator in our method (the same for each implementation for which the filter is optimized) and does not have to be the transpose of the implementation-specific backprojection operator $\boldsymbol{W}_I^T$. In order to improve stability and take additional prior knowledge of the scanned object into account, a regularization term can be added to the objective in (2.5).

The solution to the optimization problem above is a implementation-adapted filter $\boldsymbol{h}_I^*$. Once the filter has been computed, it can be used in (2.4) to give an optimized reconstruction:

$$\boldsymbol{r}_I^* = \boldsymbol{W}_I^T \boldsymbol{M}_I(\boldsymbol{h}_I^*, \boldsymbol{p}).$$

Out of all reconstructions that an implemented algorithm can produce for a given dataset $\boldsymbol{p}$ by varying the filter, this reconstruction, $\boldsymbol{r}_I^*$, is the one that results in the smallest residual error. Such filters are known as minimum-residual filters and have previously been proposed to improve reconstructions of real-space analytical algorithms in low-dose settings [14], [15].

Our implementation-adapted filters are thus minimum-residual filters that have been optimized to each implementation I. The main difference between the previous works [14], [15] and our present study is that we use a fixed forward operator in our optimization problem, which is not necessarily the transpose of the backprojection operator. More importantly, our goal in this chapter is not the improvement of reconstruction accuracy,

but the reduction of differences in reconstruction between various software implementations.

We hypothesize that such minimum-residual reconstructions obtained using different implementations are closer (quantitatively more similar) to each other than reconstructions obtained using standard filters. As an example for motivating this choice, let's take an implementation of an analytical algorithm from both TomoPy and the ASTRA toolbox. Given a certain dataset, changing the reconstruction filter results in different reconstructed images, each with a different residual error. Even though the implementations used by TomoPy and ASTRA are fixed, the freedom in choosing a filter gives us an opportunity to reduce the difference between reconstructions from both implementations. Tuning the filter is a way to *optimize* the reconstruction according to user-selected quality criteria. Choosing the *minimum-residual* reconstruction for each implementation results in reconstructions that are the *closest possible* to each other in terms of data misfit. Closeness in data misfit, under convexity assumptions, indicates closeness in pixel intensity values of reconstruction images. Hence, the minimum-residual reconstructions for the two implementations are closer to each other than reconstructions with standard filters offered by the implementations.

To compute the optimized filter (2.5), we use the fact that the reconstruction $\boldsymbol{r}_I(\boldsymbol{h}, \boldsymbol{p})$ of data $\boldsymbol{p}$ obtained from an implementation of FBP or Gridrec is *linear* in the filter $\boldsymbol{h}$. This means that we can write the reconstruction as

$$\boldsymbol{r}_I(\boldsymbol{h}, \boldsymbol{p}) = \boldsymbol{R}_I(\boldsymbol{p})\boldsymbol{h},$$

where $\boldsymbol{R}_I(\boldsymbol{p})$ is the reconstruction matrix of implementation I given projection data $\boldsymbol{p}$. Thus, the optimization problem (2.5) becomes

$$\boldsymbol{h}_I^* = \arg\min_{\boldsymbol{h}} \|\boldsymbol{p} - \boldsymbol{W}\boldsymbol{R}_I(\boldsymbol{p})\boldsymbol{h}\|_2^2 =: \arg\min_{\boldsymbol{h}} \|\boldsymbol{p} - \boldsymbol{F}_I(\boldsymbol{p})\boldsymbol{h}\|_2^2 \tag{2.6}$$

The matrix $\boldsymbol{F}_I(\boldsymbol{p})$ has dimensions $N_p \times N_f$, where N_p is the size of projection data and N_f is the number of filter components. For a filter that is independent of projection angle, the number of filter components, N_f, is equal to the number of discrete detector pixels, N_d. The projection size $N_p := N_d N_\theta$, where N_θ is the number of projection angles. $\boldsymbol{F}_I(\boldsymbol{p})$ can be constructed explicitly by assuming a basis for filter components. A canonical basis can be formed using N_d unit vectors $\{\boldsymbol{e}_i, i = 1, 2, \ldots, N_d\}$, such that

$$\boldsymbol{e}_1 = \begin{pmatrix} 1 \\ 0 \\ . \\ . \\ . \\ 0 \end{pmatrix}, \quad \boldsymbol{e}_2 = \begin{pmatrix} 0 \\ 1 \\ . \\ . \\ . \\ 0 \end{pmatrix}, \quad \ldots \quad \boldsymbol{e}_{N_d} = \begin{pmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 1 \end{pmatrix}.$$

Using these basis filters, each column of $\boldsymbol{F}_I(\boldsymbol{p})$ can be computed by reconstructing $\boldsymbol{p}$ using the implementation I, followed by forward projection with $\boldsymbol{W}$:

$$\boldsymbol{f}_j = \boldsymbol{W}\boldsymbol{r}_I(\boldsymbol{e}_j, \boldsymbol{p}), \quad j \in \{1, 2, \ldots, N_d\}$$
$$\boldsymbol{F}_I(\boldsymbol{p}) = \begin{pmatrix} \boldsymbol{f}_1 & \boldsymbol{f}_2 & \boldsymbol{f}_3 & \ldots & \boldsymbol{f}_{N_d} \end{pmatrix}$$

We can then substitute for $\boldsymbol{F}_I(\boldsymbol{p})$ in (2.6) and solve for the optimized filter $\boldsymbol{h}_I^*$. Note that our method only requires *evaluations* of the implementation I by using it as a black-box routine to compute the reconstructions $\boldsymbol{r}_I(\boldsymbol{e}_j, \boldsymbol{p})$ above. In other words, no knowledge of the implementation I or any internal coding is required.

If we expand the filter in a basis of unit vectors, $\mathcal{O}(N_p)$ reconstructions using the implementation I and $\mathcal{O}(N_p)$ forward projections with $\boldsymbol{W}$ must be performed for filter optimization. In contrast, the complexity of a standard FBP reconstruction is of the order of a single backprojection. Choosing a smaller set of suitable basis functions would result in a reduction in the number of operations for filter optimization and, consequently, faster filter computations. One way to do this is by exponential binning [14].

The idea of exponential binning is to assume that the real-space filter is a piecewise constant function with N_b bins, where $N_b < N_d$. The bin width w_i, for $i = 1, 2, \ldots, N_b$, is assumed to increase in an exponential fashion away from the centre of the detector, such that:

$$w_i = \begin{cases} 1, & |i| < N_l \\ 2^{|i|-N_l}, & |i| \geq N_l \end{cases}, \tag{2.7}$$

where N_l is the number of large bins with width 1. Exponential binning is inspired by the observation that standard filters used in tomographic reconstruction, such as the Ram-Lak filter, are peaked at the centre of the detector and decay to zero relatively quickly towards the edges. Binning results in a reduction of free filter components from N_d to N_b. Moreover, despite the reduction in components, it does not typically result in a significant change in reconstruction quality [14].

The pseudocode for our filter computation method is shown in Algorithm 1. Here we give further details of the routines used in the algorithm. The `filter` routine performs filtering in the Fourier domain, which is equivalent to multiplication by the filter followed by an inverse Fourier transform. The $\texttt{reconstruct}_I$ routine calls the function for reconstruction in implementation I with the internal filtering disabled. Finally, the `lstsq` routine calls a standard linear least squares solver in NumPy [49] to compute filter coefficients.

Algorithm 1 Implementation-adapted filter computation

procedure Compute filter($\boldsymbol{p}$, I, $\boldsymbol{W}$):
    Create filter basis: $\mathcal{B} := \{b_1, b_2, \ldots, b_{N_b}\}$
    Compute columns of $\boldsymbol{F}_I(\boldsymbol{p})$:
    for $\boldsymbol{b}_j \in \mathcal{B}$ **do**
        Filter data with basis filter: $\boldsymbol{q} \leftarrow \texttt{filter}(\boldsymbol{p}, \boldsymbol{b}_j)$
        Reconstruct filtered projection with I: $\boldsymbol{r} \leftarrow \texttt{reconstruct}_I(\boldsymbol{q})$
        Forward project reconstruction $\boldsymbol{f}_j \leftarrow \texttt{flatten}(\boldsymbol{W}\boldsymbol{r})$
    end for
    Linear least squares fitting of filter coefficients: $\boldsymbol{c} \leftarrow \texttt{lstsq}(\boldsymbol{F}_I(\boldsymbol{p}), \boldsymbol{p})$
    Return filter: $\boldsymbol{h}^* \leftarrow \sum_{j=1}^{N_b} c_j \boldsymbol{b}_j$
end procedure

Once a filter $\boldsymbol{h}^*$ is computed, we can store it in memory, either as a filter in Fourier

space or as a filter in real space after computing the Fourier transform of $\boldsymbol{h}^*$. Using the filter with a black-box software package involves calling the `filter` routine with the data and the computed filter as arguments, followed by one call of the $\texttt{reconstruct}_I$ routine in a chosen algorithm (with its internal filtering disabled). Thus, the complexity of a reconstruction using a computed implementation-adapted filter is the same as that of a reconstruction run using a standard filter.

In the following sections, we describe numerical experiments and the results of filter optimization on reconstructions.

2.4 Data and metrics

We performed a range of numerical experiments on real and simulated data to quantitatively assess (i) the effect of our proposed optimized filters on the variations between reconstructions from different implementations; (ii) the behaviour and dependence of our proposed filters on acquisition characteristics such as noise and sparse angular sampling; and (iii) the effect of our proposed filters on post-processing steps following the reconstruction block in Fig 2.1. In this section, we describe the software implementations used, data generation steps and the metric used to quantify intra-set variability of reconstructions.

2.4.1 Software implementations of analytical algorithms

We optimized filters to commonly used software implementations of FBP and Gridrec. For FBP, we considered different projector implementations in the ASTRA toolbox [43] as well as the `iradon` backprojection function in scikit-image [44]. These implementations use different choices of volume and ray discretization as well as numerical integration schemes. From the ASTRA toolbox, we considered projectors implemented on the CPU (`strip`, `line` and `linear`) as well as a pixel-driven kernel on the GPU (`gpu-pixel`, called `cuda` in the ASTRA toolbox). For Fourier-space methods, we considered the Gridrec implementation in TomoPy. We used the ASTRA `strip` kernel as the forward projector $\boldsymbol{W}$ in (2.5) during filter computations.

2.4.2 Projection data

We performed experiments with both simulated and real data. Both data consisted of projections acquired in a parallel-beam geometry along a complete angular range in $[0, \pi)$.

Simulated foam phantom data

Simulated data of foam-like phantoms were generated using the foam_ct_phantom package in Python. This package generates 3D volumes of foam-like phantoms by removing, at random, a pre-specified number of non-overlapping spheres from a cylinder of a given material [50]. The simulated phantoms are representative of real foam samples used in tomographic experiments and are challenging to reconstruct due to the presence of features at different length scales. At the same time, the phantoms are amenable to experimentation as data in different acquisition settings can be easily generated. Slices of one such

phantom, which we used for the experiments in this chapter, are shown in Fig. 2.3 and Fig. 2.5.

Ray tracing through the volume is used to generate projection data from a 3D foam phantom. To simulate real-world experimental setups, where detector pixels have a finite area, ray supersampling can be used. This amounts to averaging the contribution of n neighbouring rays within a single pixel, where n is called the supersampling factor.

For our experiments, we generated a 3D foam with 1000 non-overlapping spheres with varying radii. A parallel beam projection geometry, in line with synchrotron setups, was used to generate projection data. We used ray supersampling with a supersampling factor of 4, and each 2D projection was discretized on a pixel grid of size 256×256. We varied the number of projection angles, N_θ, in our experiments in order to determine the effect of sparse sampling ranges on our filters.

Poisson noise was added to noiseless data by using the `astra.add_noise_to_sino` function in the ASTRA toolbox [43]. This function requires the user to specify a value for the photon flux I_0. In an image corrupted with Poisson noise, each pixel intensity value k is drawn from a Poisson distribution

$$f_{\text{Pois}}(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!},$$

with $\lambda \propto I_0$. High photon counts (and high values of λ) correspond to low noise settings. All noise realizations in our experiments were generated with a pre-specified random seed.

Real data of shale

In order to validate the applicability of our method to real data, we performed numerical experiments using microCT data of the Round-Robin shale sample N1 from the tomographic data repository Tomobank [51]. We used data acquired at the Advanced Photon Source (APS) for our experiments. The Round-Robin datasets were acquired for characterizing the porosity and microstructures of shale, and the same sample has been imaged at different synchrotrons (using the same experimental settings) for comparison of results [41]. The dataset we used was acquired with a 10x objective lens and had an effective pixel size of approximately 0.7μm. Each projection in the dataset had pixel dimensions 2048×2048, and data were acquired over 1500 projection angles. In order to simulate sparse angular range settings, we removed projections at intervals of $m = 2, 3, 4, 5$ and 10 from the complete data.

2.4.3 Quantitative metrics

Reconstructions of a 3D volume from parallel beam data can be done slice-wise, because data in different slices (along the rotation axis) are independent of each other in a parallel beam geometry. Therefore, all our quantitative metrics were computed on individual slices. Reconstructed slices of the simulated foam phantom were discretized on a pixel grid of size 256×256. Reconstruction slices of the Round-Robin dataset were discretized on a pixel grid of size 2048×2048. All CPU reconstructions were performed on an Intel(R)

Core(TM) i7-8700K CPU with 12 cores. GPU reconstructions were performed on a single Nvidia GeForce GTX 1070 Ti GPU with CUDA version 10.0.

We were interested in comparing the similarity between reconstructions in a *set* of images, without having a reference reconstruction. We quantified the intra-set variability between reconstruction slices obtained from different implementations using the pixelwise standard deviation between these. For a set of reconstruction slices $\{\boldsymbol{r}_I, I \in \mathcal{I}\}$ obtained using different implementations I, the standard deviation of a pixel j is given by:

$$\sigma_j = \sqrt{\frac{1}{N_I} \sum_{I \in \mathcal{I}} \Big((r_I)_j - \bar{r}_j \Big)^2}; \qquad \bar{r}_j = \frac{1}{N_I} \sum_{I \in \mathcal{I}} (r_I)_j, \tag{2.8}$$

where $(r_I)_j$ is the intensity value of pixel j in reconstruction $\boldsymbol{r}_I$ and N_I is the total number of implementations.

In our experiments, we reconstructed the same data using our set of implementations $\{I \in \mathcal{I}\}$, by using the Ram-Lak filter and the Shepp-Logan filter as defined in different packages, and then by using filters $\{\boldsymbol{h}_I^*, I \in \mathcal{I}\}$ (2.5) that were optimized to those implementations. As a result, we achieved three sets of reconstructions: one set using the Ram-Lak filter, a second set using the Shepp-Logan filter and a third set using the implementation-adapted filters. We computed the pixelwise standard deviation (2.8) over slices for all sets.

The mean standard deviation of a slice S (with dimensions $N \times N$) is defined as the mean of pixelwise standard deviations in that slice:

$$\bar{\sigma}^S = \frac{1}{N^2} \sum_{j \in J^S} \sigma_j, \tag{2.9}$$

where J^S is the list of pixels in slice S.

In addition to the mean, the histogram of standard deviations (2.8) provides important information about the distribution of standard deviation values in a slice. The *mode* of this histogram is the value of standard deviation that occurs most, and the tail of the histogram indicates the number of large standard deviations observed. For reconstructions that are more similar to each other, we would expect the histogram to be peaked at a value close to 0 and have a small tail.

In order to quantify the difference between a reconstruction slice and the ground truth (in experiments where a ground truth was available), we used the root mean squared error (RMSE) given by

$$\text{RMSE}(\boldsymbol{r}_I) = \sqrt{\frac{1}{N^2} \sum (\boldsymbol{r}_I - \boldsymbol{r}_{gt})^2}, \tag{2.10}$$

where $\boldsymbol{r}_{gt}$ is the ground truth reconstruction. For a set of reconstructions we used the squared bias defined below to quantify the difference with respect to the ground truth:

$$\Big(\text{bias}(\{\boldsymbol{r}_I, I \in \mathcal{I}\}) \Big)^2 = \Big(\bar{\boldsymbol{r}} - \boldsymbol{r}_{gt} \Big)^2, \tag{2.11}$$

where $\bar{\boldsymbol{r}} := \sum_{I \in \mathcal{I}} \frac{1}{N_I} \boldsymbol{r}_I$ is the mean over the set of reconstructions. The squared bias, similar to the standard deviation in (2.8) is a pixelwise measure. The mean squared bias over a slice S is obtained by taking the mean of (2.11) over all pixels in the slice.

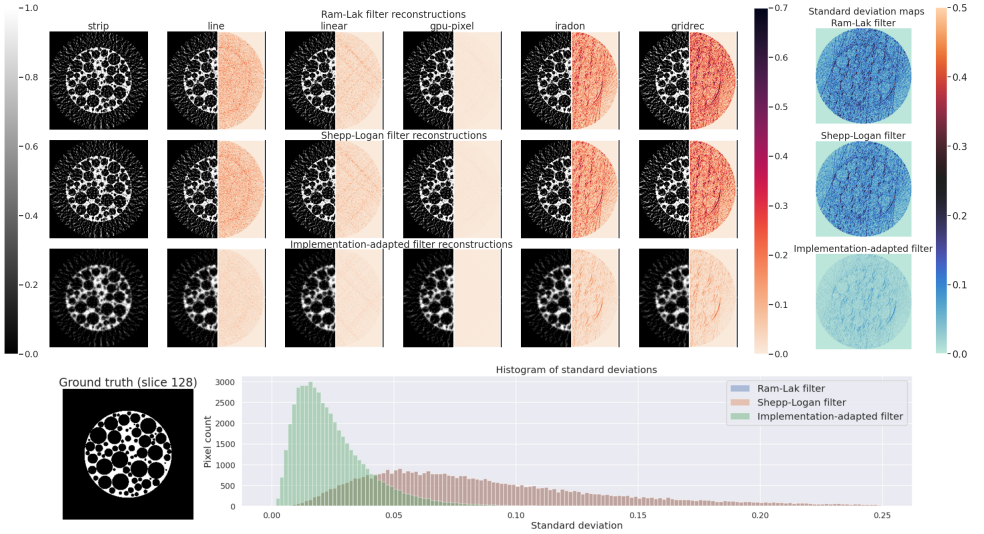

Figure 2.3: Reduction in intra-set variability between reconstructions of simulated foam data ($N_\theta = 32$, no noise) by using implementation-adapted filters. (*top three rows*) Reconstructions of the central slice (slice no. 128) of a foam phantom. To highlight intra-set discrepancies we show the absolute difference with respect to the corresponding `strip` kernel reconstructions in the right half of each image. The rightmost column shows pixelwise standard deviation σ in each set. (*bottom row, left*) Ground truth foam phantom slice. (*right*) Histograms of standard deviations σ for all three sets. The Ram-Lak filter and Shepp-Logan filter histograms overlap.

In our experiments, we also quantify the effect of filter optimization on later post-processing steps after reconstruction. To do this, we threshold a set of reconstructions using Otsu's method [52], which picks a single threshold to maximize the variance in intensity between binary classes. To quantify the accuracy of the resulting segmentations and to compare the similarity in a set we used two standard metrics for segmentation analysis: the F_1 score and the Jaccard index. The F_1 score takes into account false positives (fp), true positives (tp) and false negatives (fn) in binary segmentation and is given by:

$$F_1 = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}. \tag{2.12}$$

The Jaccard index is the ratio between the intersection and union of two sets A and B. In our case, one set is the segmented binary image and the other set is the binary ground truth image:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{2.13}$$

2.5 Numerical experiments and results

In this section, we give details of our numerical experiments and discuss their results.

2.5.1 Foam phantom data

Reduction in differences between reconstructions

Fig. 2.3 shows the central (ground truth) slice of the foam phantom. Data along $N_\theta = 32$ angles were reconstructed using all implementations using the Ram-Lak filter, the Shepp-Logan filter and our implementation-adapted filters. Reconstructions using the various filters are shown in Fig. 2.3. In order to highlight intra-set variability, we include heatmaps showing the absolute difference with respect to one (`strip`) reconstruction. Upon visual inspection, we see that discrepancies between reconstructions are smaller in the set obtained using implementation-adapted filters. An interesting point to note is that the Gridrec and `iradon` reconstructions show the largest differences from the ASTRA `strip` kernel reconstruction in both sets. This suggests that differences between different software packages are greater than differences between different projectors in the same software package.

To further investigate intra-set variability, we use pixelwise standard deviation maps for all sets of reconstructions. We observe that higher values of standard deviation are observed when using the Ram-Lak and Shepp-Logan filters. This indicates that quantitative differences between these reconstructions were more pronounced. In contrast, reconstructions using our implementation-adapted filters were more similar, resulting in low pixelwise standard deviations. Furthermore, the mode of the histogram of standard deviations (in the central slice) is shifted closer to zero for reconstructions with our filters, and the tail of the histogram is shorter. This highlights the fact that the *maximum* standard deviation between reconstructions with our filters is smaller than the maximum standard deviation in reconstructions with the Shepp-Logan or Ram-Lak filters.

Dependence of filters on noise and sparse angular sampling

We consider the effect of noise and sparse sampling on our filters. For the central slice of the foam phantom shown in Fig. 2.3, we generated data by varying the number of projection angles N_θ and the photon flux I_0. For each of these settings, we computed the mean standard deviation (2.9) between reconstruction slices. Our results are shown in Fig. 2.4. For all noise and angular sampling settings, the mean standard deviation in the slice was reduced by using implementation-adapted filters, with the difference being particularly prominent for noisy and smaller angular sampling settings. Shepp-Logan filter reconstructions had smaller mean standard deviation compared with Ram-Lak filter reconstructions, except in situations where many angles ($N_\theta \geq 256$) were used. In the high angle regime, reconstructions using the Ram-Lak filter have a relatively small number of artefacts and improvements due to filter optimization are modest.

We also quantified the mean squared bias and the mean RMSE with respect to the ground truth for this slice. From these plots, we observe that reconstructions using implementation-adapted filters have lower mean squared bias and mean RMSE compared

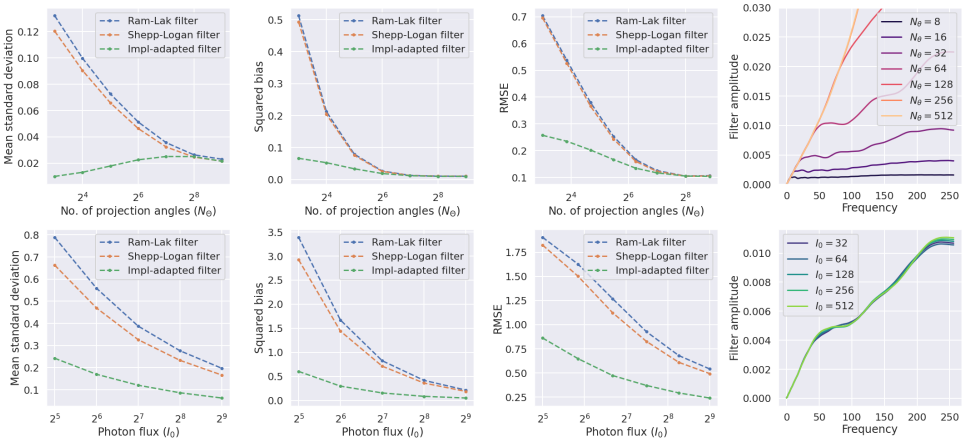

Figure 2.4: Implementation-adapted filters for noisy and sparsely sampled data. (*top, left to right*) Mean standard deviations $\bar{\sigma}^S$ for slice $S = 128$ as a function of the number of projection angles N_θ, mean value of the squared bias, mean value of RMSE with respect to the ground truth slice, and optimized filters in Fourier space. (*bottom, left to right*) Mean standard deviations in $S = 128$ as a function of photon flux I_0 (higher values of I_0 correspond to lower noise levels) using $N_\theta = 64$, mean value of the squared bias, mean value of RMSE with respect to the ground truth slice, and optimized filters in Fourier space.

with those for reconstructions with standard filters. High noise (low I_0) and sparse angular sampling settings result in an increase in bias and RMSE for all filter types. However, the increase is sharper for the Shepp-Logan and Ram-Lak filters than for our implementation-adapted filters. For every noise setting, the Ram-Lak filter results in the worst reconstructions in terms of bias and RMSE. Although both bias and RMSE increase as the number of projection angles is reduced in the noise-free setting, we observe a reduction in mean standard deviation for reconstructions using implementation-adapted filters. This suggests that in spite of a reduction in mean standard deviation due to effective suppression of high frequencies, the reconstructions produced by our implementation-adapted filters in this regime are incapable of mitigating the large number of low-angle artefacts. In effect, these settings show a limit where optimization of a linear filter is not sufficient for good reconstructions, and intra-set homogeneity is achieved at the expense of an increase in bias and RMSE.

In addition, we also show the shapes of the filters (computed for the `strip` kernel in the ASTRA toolbox) as a function of noise and angular sampling. As the number of projection angles is increased, the shape of implementation-adapted filters approaches that of the ramp filter. In these regimes, reconstructions obtained using the Ram-Lak filter and the Shepp-Logan filter are nearly identical in terms of bias and RMSE. For different noise settings, the filters only vary at certain frequencies. It is possible that these frequencies are indicative of the main features in the foam phantom slice used.

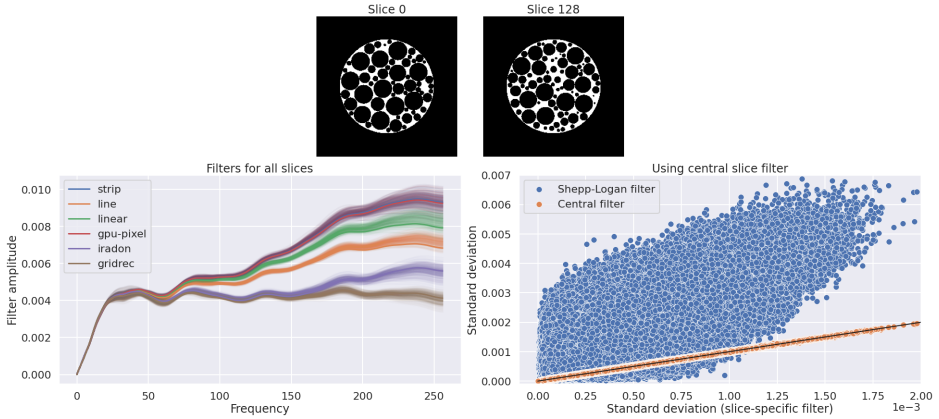

Figure 2.5: Variation of filters with projection data. (*top*) Two slices of a simulated foam phantom with differences in features. (*bottom left*) Implementation-adapted filters for all slices of the foam phantom (slice-specific filters). Central slice (slice no. 128) filters for each implementation are indicated with bold lines. (*bottom right*) Scatter plot of pixelwise standard deviations σ using slice-specific filters, the central slice filter and the Shepp-Logan filter. Standard deviations using the central slice filter are almost the same as those using slice-specific filters (orange dots). These points lie on a straight line (shown in black) with slope ~ 1 and intercept ~ 0. In contrast, standard deviations using the Shepp-Logan filter are higher than those using slice-specific filters (blue dots) for most pixels.

Variation of filters with projection data

In order to understand how our filters change with changes in the data, we computed filters for all slices of our simulated foam phantom. Two such slices are shown in Fig. 2.5. These slices, although visually similar, have different features. Implementation-adapted filters for all 256 slices of the foam phantom are shown in Fig. 2.5.

In order to study the applicability of the central slice filter to other slices, we performed the following experiment. First, we reconstructed all slices using the slice-specific filters, i.e. filters that had been optimized for *each individual slice* using different implementations. Next, we reconstructed all slices with the central slice filter. As a baseline, all slices were also reconstructed using the Shepp-Logan filter. Pixelwise standard deviations (2.8) were computed for all pixels in the foam phantom volume for the three cases. The scatter plot in Fig. 2.5 shows that the pixelwise standard deviations with the central slice filter are nearly the same as those with the slice-specific filters. In fact, these points lie on a line with slope nearly equal to one. This indicates that using the central slice filter results in an equivalent reduction in differences between reconstructions as slice-specific filters. In contrast, the pixelwise standard deviations using the Shepp-Logan filter are, for a majority of pixels, larger than those obtained using slice-specific filters. This suggests that, for a majority of pixels in the reconstruction volume, smaller values of standard deviation are observed after filter optimization.

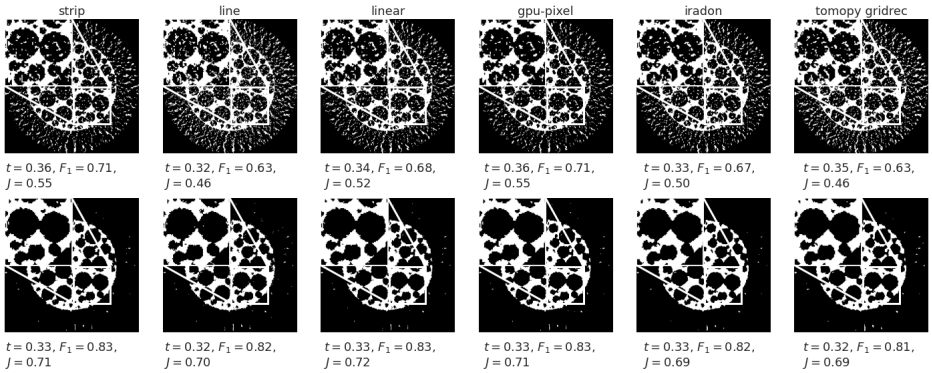

Figure 2.6: Differences after thresholding using Otsu's method. Reconstructions shown in Fig. 2.3 were used as input to the thresholding routine. (*top row*) Thresholded reconstructions obtained using different backprojector implementations and the Shepp-Logan filter. Corresponding Otsu thresholds t, F_1 scores and Jaccard indices are given for each image. (*bottom row*) Thresholded reconstructions obtained using implementation-adapted filters.

Our experiment suggests that using the central slice filter for all slices of the foam phantom results in an equivalent reduction in standard deviation as slice-specific filters. This paves the way to fast application of such filters in a real dataset. An implementation-adapted filter computed for one slice of such a dataset could be reused with all other slices with no additional computational cost, just like any of the standard filters in a software package.

Reduction in differences after thresholding

We investigated the effect of our filters on the results of a simple post-processing step. We reconstructed data ($N_\theta = 32$, no noise) from the central slice of the foam phantom and used Otsu's method in `scikit-image` [44] to threshold reconstruction slices from different implementations. In Fig. 2.6, we show two sets of thresholded reconstructions, one obtained using the Shepp-Logan filter and the other obtained using our implementation-adapted filters. We show values for the Otsu threshold t, the F_1 score with respect to the ground truth slice and the Jaccard index in the figure. We used routines in `scikit-learn` [53] to compute all segmentation metrics. For the set of Shepp-Logan filter reconstructions, the ranges of threshold values (0.32-0.36), F_1 scores (0.63-0.71) and Jaccard indices (0.46-0.55) were larger than the corresponding ranges for the implementation-adapted filter reconstructions. For the latter set, the Otsu threshold varied between 0.32 and 0.33 for all reconstructions. The F_1 scores were between 0.81 and 0.83, and the Jaccard indices were in the range of 0.69-0.72. Upon visual inspection of the zoomed-in insets we find greater differences between thresholded reconstructions in the set of Shepp-Logan filter reconstructions. These results suggest that post-processing steps such as segmentation may be rendered more reproducible and amenable to automation if reconstructions are obtained using implementation-adapted filters.

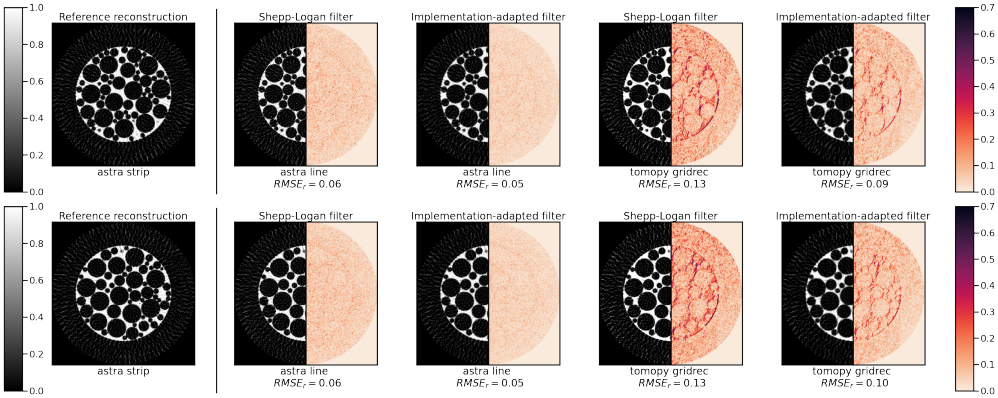

Figure 2.7: Filter optimization using a reference reconstruction. (*top row*) Filters optimized to a `strip` kernel reconstruction (*top row, left*). (*top row*) Reconstructions before and after filter optimization using the ASTRA `line` kernel and Gridrec. Right half of each image shows absolute difference with the reference reconstruction. RMSE values with respect to the reference are also shown. (*bottom row*) Reconstructions of a different (test) slice using the filters obtained for the slice in the top row. Pixelwise absolute difference and RMSE using implementation-adapted filters are smaller in both cases.

Optimizing to a reference reconstruction

Although we focus on filter optimization in sinogram space in this chapter, a related optimization problem is one where reconstruction results from different implementations are optimized to a reference reconstruction. This type of optimization might be useful when the result of one specific implementation is preferred due to its superior accuracy and when the exact settings used with this algorithm are unknown.

In some cases, high-quality reconstructions might be computed with an unknown (possibly in-house) software package during the experiment by expert beamline scientists. When users reconstruct this data later at their home institutes, it might not be possible to use the same software packages with identical settings. Our approach would enable users to reduce the difference between their reconstructions and the high-quality reference reconstructions.

Optimization in reconstruction space can be performed by modifying the objective in (2.5):

$$\boldsymbol{h}_I^* = \arg\min_{\boldsymbol{h}} \|\boldsymbol{r}_{\text{ref}} - \boldsymbol{r}_I(\boldsymbol{h}, \boldsymbol{p})\|_2^2, \tag{2.14}$$

where $\boldsymbol{r}_{\text{ref}}$ is the reference reconstruction.

To illustrate filter optimization in reconstruction space, we performed the following experiment. Using the `strip` kernel reconstruction (with the Shepp-Logan filter) as a reference, we computed optimized filters for two other implementations (ASTRA `line` kernel and TomoPy Gridrec) for reconstructing the central slice of the foam phantom. Subsequently, we reconstructed the sinogram with the Shepp-Logan filter and our filters. These reconstructions are shown in the top row of Fig. 2.7. To quantify similarity with

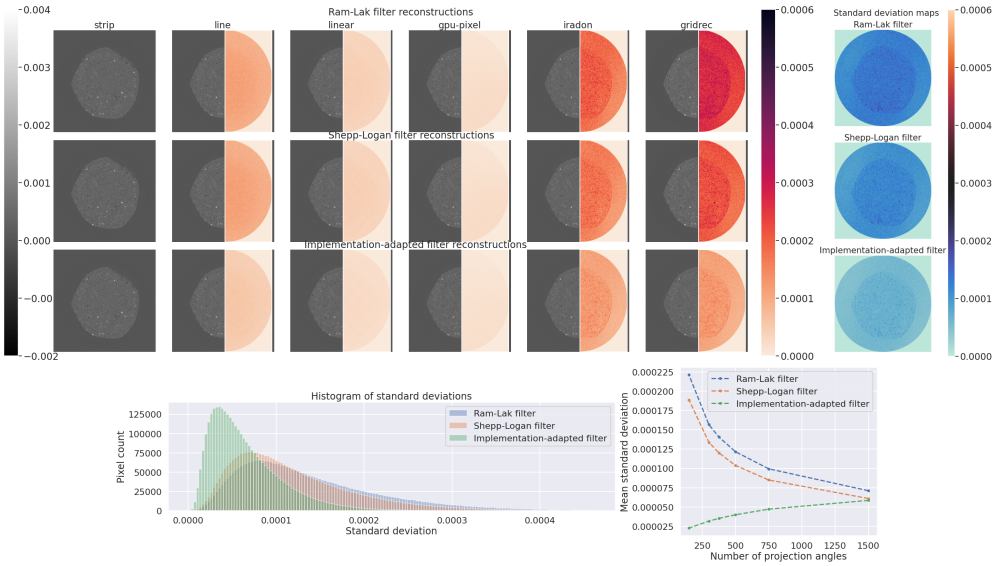

Figure 2.8: Reduction in differences between reconstructions of the Round-Robin dataset (slice no. 896). (*top three rows*) Slice reconstructions using different implementations. Reconstructions were performed by discarding every second projection from the full dataset. The right half of the images show absolute differences with the corresponding `strip` kernel reconstruction in each set. The rightmost column shows pixelwise standard deviations in each set. (*bottom row, left*) Histograms of standard deviation for all three types of filters. (*right*) Mean standard deviations $\bar{\sigma}^S$ in slice $S = 896$ for different numbers of projection angles.

the reference reconstruction, we computed the pixelwise absolute difference between each reconstruction and the reference as well as the RMSE using the reference as ground truth, which we denote as RMSE_r. For both `line` and Gridrec backprojectors, optimizing the filter to a reference reconstruction reduced the RMSE_r and absolute difference. As a further test, we applied the filters computed for this slice to a different slice of the foam phantom, which did not have any overlaps with the slice used to compute the filters. For this test slice, we again observed the reduction in RMSE_r and absolute error, suggesting that our filters were able to bring the resulting reconstructions closer to the reference reconstruction.

2.5.2 Round-Robin data

Fig. 2.8 shows the results of our method on the central slice (slice no. 896) of the Round-Robin dataset N1. These reconstructions were performed by discarding every second projection from the entire dataset. From the heatmaps of absolute difference with respect to the `strip` kernel reconstruction, we observe that intra-set differences are reduced by using implementation-adapted filters. This is further shown by the pixelwise standard

deviation maps. Standard deviations between reconstructions using the Ram-Lak and Shepp-Logan filters are larger than those between reconstructions using implementation-adapted filters. Similar to the distributions in Fig. 2.3, we see that our implementation-adapted filters are able to shift the mode of the histogram of standard deviations towards zero and to reduce the number of large standard deviations in the slice. We also observe that the Ram-Lak filter reconstructions show higher standard deviations than the Shepp-Logan filter reconstructions.

We also studied the effect of the number of projections used on the mean standard deviation (2.9) in this slice. To do this, we performed experiments with the whole dataset and also with parts of the data, where every $2, 3, 4, 5$ and 10 projections were discarded. For each instance, the data were reconstructed using the Ram-Lak filter, the Shepp-Logan filter and our implementation-adapted filters. The plot of mean standard deviations is shown in Fig. 2.8. For all projection numbers, filter optimization reduced the mean standard deviation in the slice. The difference was smaller for higher projection numbers, indicating that our filters are especially useful in improving reproducibility of reconstructions when the number of projection angles is small. In practice, data along few angles may be acquired to reduce the X-ray dose on a sample or to speed up acquisition when the sample is evolving over time.

2.6 Discussion

In this chapter, we presented a method to improve the reproducibility of reconstructions in the synchrotron pipeline. Our method uses an optimization problem over filters to reduce differences between reconstructions from various software implementations of commonly-used algorithms.

The objective function that was used in our optimization problem was the ℓ^2-distance between the forward projection of the obtained reconstruction and the given projection data. This choice was motivated by the fact that ground truth reconstructions are generally not available in real-world experiments. However, it is possible to formulate a similar (and related) problem in reconstruction space, by using the ℓ^2-distance between the reconstruction from a given software package and a reference reconstruction as the objective to be minimized. The solution to such an optimization procedure is a shift-invariant blurring kernel in reconstruction space. The implementation-adapted filters presented in this chapter can thus be viewed as a linear transformation of the projection data that results in an automatic selection of shift-invariant blurring of reconstructions.

Our work here can be extended to optimize other pre-processing and post-processing steps in the synchrotron pipeline. An important example is phase retrieval, which can be formulated in terms of a filtering operation [35]. This filter can be optimized similarly in order to improve reproducibility.

One limitation of our method is that we optimize to the data available. This optimization can lead to undesired solutions in the presence of outliers in the data, such as zingers or ring artefacts. Reconstructions of data corrupted with zingers (randomly placed very bright pixels in the sinogram) are shown in Fig. 2.9. In this example we see that the FBP reconstruction using the ASTRA `strip` kernel and the Shepp-Logan filter shows less

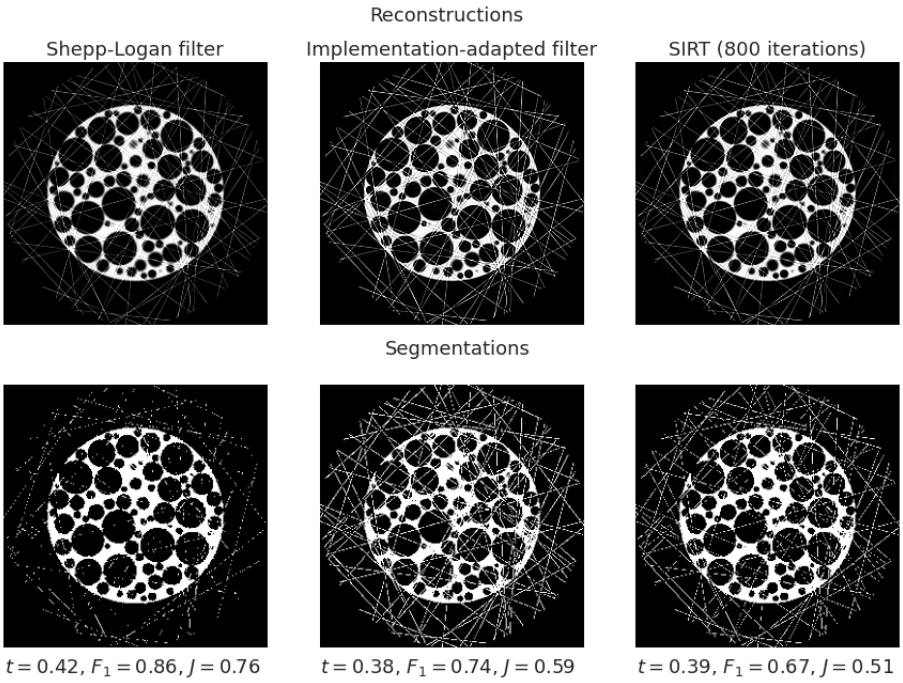

Figure 2.9: Reconstructions of data corrupted with zingers showing an example where the Shepp-Logan filter reconstruction and corresponding segmentation are better than those using an implementation-adapted filter or an iterative method (SIRT). (*top row*) Reconstructions of data from slice 128 ($N_\theta = 512$, no noise) corrupted with zingers. Zingers are more prominent in the reconstruction using an implementation-adapted filter and in the SIRT reconstruction (after 800 iterations). (*bottom row*) Segmentations using Otsu's method of all three reconstructions. The Otsu threshold, F_1 score and Jaccard index for each image is given below.

prominent zingers than the reconstruction using an implementation-adapted filter. This is because the optimized filter preserves the zingers in the data whereas the unoptimized FBP reconstruction is independent of them. Other methods, such as the simultaneous iterative reconstruction technique (SIRT), which iteratively minimize the data misfit also give similar, poor reconstructions. One way to improve iterative reconstruction methods is to use regularization, which can be achieved either by early stopping or by the inclusion of an explicit regularization term in the objective function to be minimized. Analogous techniques can be used for our filter optimization problem (2.5) to ensure greater robustness to outliers.

Although we have demonstrated the reusability of our filters for similar data, these filters are dependent on the noise statistics and angular sampling in the acquired projections. One way to improve the generalizability of filters would be to simultaneously optimize to more than one dataset. This idea has been explored in [54], [55] using shallow neural

networks.

Another promising direction is provided by deep learning-based methods, which have been applied to improve tomographic image reconstruction in a number of ways [56]. Supervised deep learning approaches can be used to learn a (non-linear) mapping from input reconstructions to a reference reconstruction. However, such approaches generally require large amounts of paired training data (input and reference reconstructions). When insufficient training pairs are available, various unsupervised approaches, such as the Deep Image Prior method proposed in [57], are more suitable. For a quantitative comparison of various popular deep learning-based reconstruction methods, we refer the reader to [58].

Apart from software solutions for image reconstruction, which have been the focus of this chapter, improving reproducibility throughout the synchrotron pipeline requires hardware adjustments to the blocks in Fig 2.1. Scanning the same sample twice under the same experimental conditions leads to small fluctuations in the data due to stochastic noise and drifts during the scanning process. In addition, beam-sensitive samples might deform due to irradiation. Such changes lead to differences in reconstructions that are similar to the differences due to software implementations, albeit less structured than those shown in Fig. 2.2. To improve hardware reproducibility, controlled phantom experiments might be performed to address differences in data acquisition. Finally, software and hardware solutions can be effectively linked by using approaches like reinforcement learning for experimental design and control [59], [60]. Such creative solutions might provide an efficient way for synchrotron users to perform reproducible experiments in the future.

2.7 Conclusion

In this chapter, we proposed a filter optimization method to improve reproducibility of tomographic reconstructions at synchrotrons. These implementation-adapted filters can be computed for any black-box software implementation by using only evaluations of the corresponding reconstruction routine. We numerically demonstrated the properties of and use cases for such filters. In both real and simulated data, our implementation-adapted filters reduced the standard deviation between reconstructions from various software implementations of reconstruction algorithms. The reduction in standard deviation was especially evident when the data were noisy or sparsely sampled.

Our filter optimization technique can be used to reduce the effect of differences in discretization and interpolation in commonly-used software packages and is a key building block towards improving reproducibility throughout the synchrotron pipeline. We make available the open-source Python code for our method, allowing synchrotron users to obtain reconstructions that are more comparable and reproducible.