LEAP Technical Manual

Kyle Champley

November 27, 2023

1 Introduction

This document serves as a technical description of the algorithms in LivermorE AI Projector (LEAP) library. LEAP is a C/C++/CUDA library of tomographic projectors (forward and back projection) implemented for both multi-GPU and multi-core CPU. We provide bindings to PyTorch to achieve differentiable forward and backward projectors for AI/ML-driven Computed Tomography (CT) applications.

Our projectors are implemented for the standard 3D CT geometry types: parallel-, fan-, and cone-beam. These geometry types accommodate shifts of the detectors and non-uniform angular spacing. Cartoonized sketches of these CT geometry types is shown in Figure 1. For added flexibility, we also provide a flexible modular-beam format where the user may specify the location and orientation of every source and detector pair. These projectors use the Separable Footprint method [Long, Fessler, and Balter, TMI, 2010] which provides a matched projector pair that models the finite size of the voxel and detector pixel. These matched projectors ensure convergence and provide accurate, smooth results. Unmatched projectors or those projectors that do not model the finite size of the voxel or detector pixel may produce artifacts when used over enough iterations [DeMan and Basu, PMB, 2004].

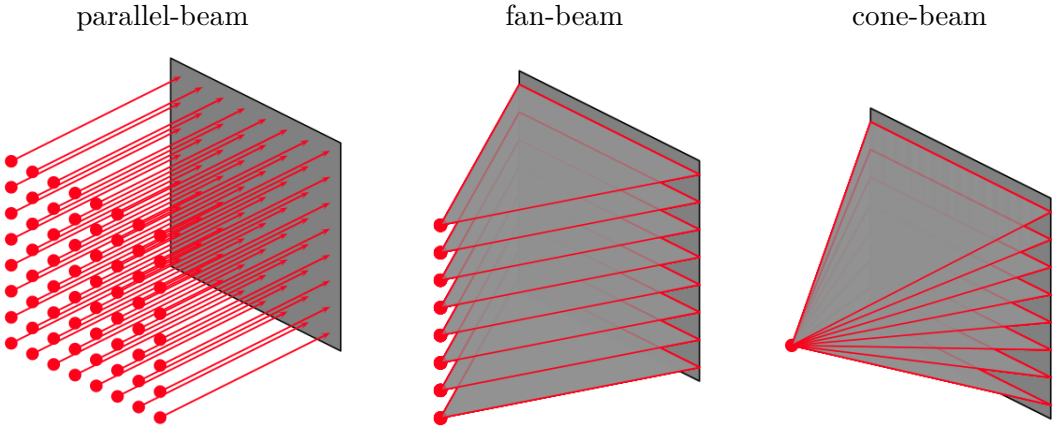

Figure 1: Cartoon sketches of the parallel-, fan-, and cone-beam geometries.

We also provide projectors and analytic inversion algorithms, i.e., FBP, for a few specialized x-ray/Radon transforms:

1. Cylindrically-symmetric/anitsymmetric objects (related to the Abel Transform) in parallel- and cone-beam geometries with user-specified symmetry axis [Champley and Maddox, Optica, 2021]. These are often used in flash radiography applications.

2. Attenuated Radon Transform (ART) for parallel-beam geometries. These are used in parallel-hole collimator SPECT and Volumetric Additive Manufacturing (VAM).

In addition to the projectors, we also provide a few other algorithms for tomographic imaging, including:

1. Quantitatively-accurate analytic inversion algorithms, i.e., Filtered Backprojection (FBP) for each geometry except modular-beam.

2. A GPU implementation of 3D anisotropic Total Variation (TV) functional, gradient, and quadratic form to be used in regularized reconstruction.

3. Python implementations of some iterative reconstruction algorithms: OSEM, SART, and RWLS.

The CPU- and GPU-based projectors are nearly identical (32-bit floating point precision) and are quantitatively accurate and thus can be used in conjuction with physics-based corrections, such as, scatter and beam hardening correction. If one is looking for a more general-purpose and full functioning CT software package (it does not, however, work with PyTorch and is closed-source), see LTT [**?**].

2 Software Architecture

The majority of LEAP is written in C/C++/CUDA using the standard libraries. The main class is called *tomographicModels*. All calls to LEAP must go through this class. This class contains functions to call the projectors (forward projection and backprojection), FBP reconstruction algorithms, and noise filter functions. It also contains a class called *parameters* which contains all the CT geometry, CT volume parameters, and other run-time parameters.

There are two different interfaces to *tomographicModels* class. The first interface uses torch and PyTorch, so that LEAP can be used as differentiable projectors in ML/DL applications. Installation of this interface is achieved with a pip install command. The second interface uses standard ANSI C and a simple ctypes Python binding. This interface is for those that want to use LEAP as a library of tomographic functions and want a version of LEAP that does not require torch/PyTorch. Installation of this interface is achieved with cmake. Installation instructions are given in the github wiki page. A diagram of the LEAP software architecture is shown in Figure 2.

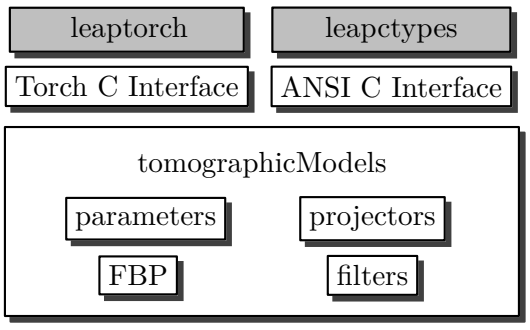

Figure 2: Overview of LEAP software architecture. The gray boxes are Python and the white boxes are C/C++/CUDA.

3 Troubleshooting Tools

LEAP has a fair amount of parameters to specify and its algorithms are complex. To assist users in troubleshooting, we have developed several tools. The first tool simply prints all the parameter settings to the screen. The function is given by

printParameters().

The second tool builds a 3D sketch (using matplotlib) of the CT geometry and the CT volume. This sketch can be generated by

sketchSystem().

An example of the output of this function with a modular-beam geometry is shown in Figure 4.

4 LEAP Parameters and General Syntax

LEAP is a library of CT algorithms that operates on data arrays provided by the user. These data arrays are the projection data and the volume data (see Section 4.3). The only permanent memory that LEAP manages is a class with member variables that parameterize the CT geometry and CT volume. Some temporary memory is allocated during the computation of an algorithm but it is destroyed before the completion of the algorithm. Prior to running any algorithm in LEAP, one must first set the CT geometry parameters and the CT volume parameters.

4.1 Setting the CT Geometry Parameters

Setting the CT geometry parameters is done by running one of the functions below

set_parallelBeam(numAngles, numRows, numCols, pixelHeight, pixelWidth, centerRow, centerCol, phis)

set_fanBeam(numAngles, numRows, numCols, pixelHeight, pixelWidth, centerRow, centerCol, phis, sod, sdd)

set_coneBeam(numAngles, numRows, numCols, pixelHeight, pixelWidth, centerRow, centerCol, phis, sod, sdd)

set_modularBeam(numAngles, numRows, numCols, pixelHeight, pixelWidth, sourcePositions, detectorCenters, rowVecs, colVecs)

A description of these parameters can be found in Tables 1 and 2.

To use the cylindrically-symmetric projectors, one must set the *axisOfSymmetry* parameter. This is done using the following function

set_axisOfSymmetry(angleInDegrees)

which must be a number between -30 and 30 degrees. This axis of symmetry is nominally the z-axis. The *axisOfSymmetry* parameter specifies the angle of rotation of the axis of symmetry around the x-axis. One must specify numAngles = 1 for this to work.

The Attenuated Radon Transform (ART) is used to model parallel-hole SPECT data and its adjoint can be used as a model in Volumetric Additive Manufacturing (VAM). In these contexts, the *CT volume* in LEAP, plays the role as the activity concentration in units of Bq/mm^3. To use the ART projectors, one must set the attenuation map and the CT geometry must be parallel-beam. One can either specify this map as a voxelized volume (which must be defined on the same voxel grid as the standard volume and its units are in inverse length, e.g., mm^{-1}) or as a constant-valued function defined on a cylinder. One can use the follow two functions to set this attenuation map.

Table 1: Parallel-, fan-, and cone-beam parameters. Note that sod and sdd only apply to fan- and cone-beam geometries and the centerRow parameter has no effect on parallel- and fan-beam geometries. Note that the phis parameter values *must* be monotomic. Internally, LEAP maintains a copy of the phis array, so if you want to make changes to it, you must either re-specify all parameters or use the set_angles function. More information can be found in Section 7.

Parameter Name	Type	Description
numAngles	int32	number of projection angles
numRows	int32	number of detector rows
numCols	int32	number of detector columns
pixelHeight	float32	detector pixel pitch between rows (mm)
pixelWidth	float32	detector pixel pitch between columns (mm)
centerRow	float32	detector pixel row index for the ray that passes from the source, through the origin, and hits the detector
centerCol	float32	detector pixel column index for the ray that passes from the source, through the origin, and hits the detector
phis	float32 array	array of the projection angles (degrees)
sod	float32	source to object distance (mm)
sdd	float32	source to detector distance (mm)

Table 2: Modular-beam parameters. Internally, LEAP maintains copies of sourcePositions, detectorCenters, rowVecs, and colVecs, so if you want to make changes to these, you must re-specify the modular-beam parameters. More information can be found in Section 7.

Parameter Name	Type	Description
numAngles	int32	number of projection angles
numRows	int32	number of detector rows
numCols	int32	number of detector columns
pixelHeight	float32	detector pixel pitch between rows (mm)
pixelWidth	float32	detector pixel pitch between columns (mm)
sourcePositions	float32 array	numAngles $\times$ 3 array of (x,y,z) coordinates of each source position
detectorCenters	float32 array	numAngles $\times$ 3 array of (x,y,z) coordinates of each detector center position
rowVecs	float32 array	numAngles $\times$ 3 array of vectors pointing in the positive detector row direction
colVecs	float32 array	numAngles $\times$ 3 array of vectors pointing in the positive detector row direction

set_attenuationMap(3D array)

set_cylindircalAttenuationMap(attenuationCoefficient, attenuationRadius)

If using the second of these two functions, the attenuationCoefficient must be floating point number whose units are in inverse length (e.g., mm^{-1}) and the attenuationRadius must be a floating point number listed in length units (e.g., mm). Only GPU projectors are defined for the arbitrary attenuation map. The constant-valued attenuation map work for both CPU and GPU processing and is much faster. Analytic inversion algorithms are carried out via Novikov's formula and must have at least 360 degrees of angular coverage.

4.2 Setting the CT Volume Parameters

Next, the user should specify the CT volume parameters. This can be done in one of the two following ways:

set_volume(numX, numY, numZ, voxelWidth, voxelHeight, offsetX, offsetY, offsetZ)

set_defaultVolume()

Using the set_defaultVolume() function (after the CT geometry parameters are set) sets the CT volume parameters to fill the field of view of the CT system with the nominal voxel sizes. A description of these parameters can be found in Table 3.

Table 3: CT volume parameters. A full description of these parameters can be found in Section 7.2.

Parameter Name	Type	Description
numX	int32	number of voxels in the x-dimension
numY	int32	number of voxels in the y-dimension
numZ	int32	number of voxels in the z-dimension
voxelWidth	float32	voxel pitch (size) in the x and y dimensions (mm)
voxelHeight	float32	voxel pitch (size) in the z dimension (mm)
offsetX	float32	shift the volume in the x-dimension (mm)
offsetY	float32	shift the volume in the y-dimension (mm)
offsetZ	float32	shift the volume in the z-dimension (mm)

4.3 Layout of the Data Arrays

The data arrays must be contiguous. The order of the projection data indices is listed as follows

projection_data[iAngle*numRows*numCols + iRow*numCols + iCol],

where iAngle, iRow, and iCol are the indices of the angle, row, and column dimensions. The volume data can be in ZYX or XYZ order. The ZYX order is the default setting. To switch to the XYZ ordering do: set_volumeDimensionOrder(0). The ZYX ordering works best for GPU processing (and is *required* for multi-GPU processing) and the XYZ ordering works best for CPU processing. To be clear the ZYX ordering is as follows

volume_data[iZ*numY*numX + iY*numX + iX]

and the XYZ ordering is as follows

volume_data[iX*numY*numZ + iY*numZ + iZ],

where iX, iY, and iZ are the indices of the x, y, and z dimensions. To help assist users in allocating contiguous arrays in the correct order in Python we provide the follow Python functions

allocateProjections()

allocateVolume()

5 CPU/GPU Processing

As stated above, the algorithm in LEAP are written for both multi-GPU and multi-core CPU processing. If one has at least one GPU, then GPU is the default processing.

There is an integer array parameter in LEAP where you specify which GPUs you would like to use. This parameter can be set with the function

set_GPUs(list of integers).

GPUs are listed on a computer as integers start at zero. The first number in this list is considered the primary GPU. For small computations it is advantageous to only use one GPU and not transfer any data on/off this GPU. Thus if the data is already on the GPU, LEAP will only use this GPU for processing. On the other hand, if the data is on the CPU memory, then the data will be transferred to each GPU in your list, the computation is carried out and then the result is copied back to the CPU array you specified.

For convenience, we also provide the function: set_GPU(gpu index) to only set a single GPU. To switch to CPU processing, run this command: set_GPU(-1).

6 A Note on Coordinate Systems

LEAP uses the standard right-hand coordinate system, but there are important subtleties in CT coordinate systems so we thought we'd start off by describing these. Typical two-dimensional Cartesian coordinate systems have the positive x-coordinate pointing to the right and the positive y-coordinate pointing up. On the other hand, typical digital image coordinates have the column coordinate pointing to the right and the row coordinate pointing *down*. This discrepancy between digital image coordinates and Cartesian coordinates causes endless confusion. Images are displaced on a computer screen this way and digital x-ray detectors record data this way as well. Thus, in LEAP, we make the effort to use the digital image coordinate system as well as we can.

Thus, we define detector coordinates as digital image coordinates with the origin at the top-left side of the image/detector. So that out Cartesian coordinates more closely match this convention, our z-axis shall be parallel to the detector row coordinate (and point in the same direction) and our x-axis shall be parallel to the detector column coordinate (and point in the same direction). The y-axis shall be oriented to make a right-hand coordinate system. A sketch of these coordinates in the cone-beam geometry is shown in Figure 3.

7 CT Geometries

In this section we discuss the parameterization of the four LEAP scanner geometry types: parallel-, fan-, cone-, and modular-beam. Cartoon sketches of the parallel-, fan-, and cone-beam geometries is shown in Figure 1 and a detailed sketch of the cone-beam coordinates in Figure 3. Except for the modular beam data set, we also give the mathematical expressions that link sets of parameterizations, state conjugate ray expressions, and the adjoint of the forward projection operators which are often refered to as backprojection operators. Unless otherwise stated, all LEAP algorithms can be applied to any scanner geometry type. This feature is made possible by the fact that data for all geometry types can be stored in contiguous 3D arrays (view number, detector row, detector column) and that most of the differences can be encapsulated in the implementation of the forward and back projectors. Two-dimensional geometries can be achieved by setting the number of detector rows and number of z-slices to one.

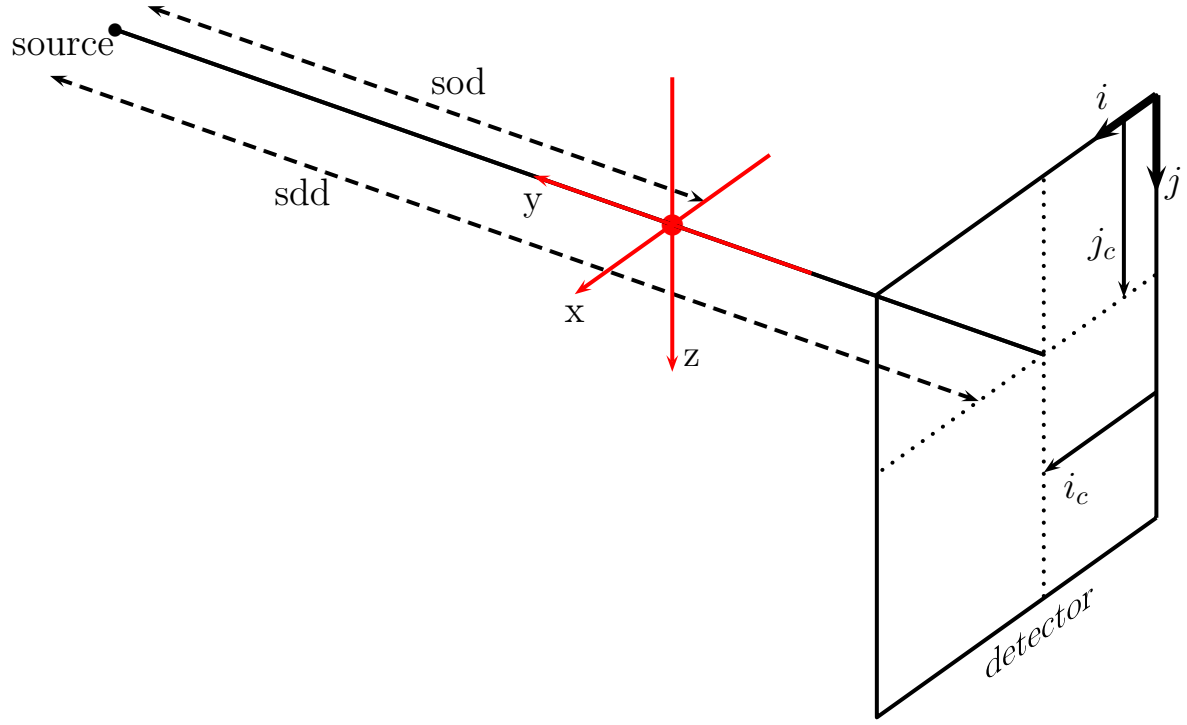

Figure 3: LEAP provides forward and backprojectors and FBP reconstruction algorithms for a flexible cone-beam geometry. Note that (i, j) are detector column and row index coordinates, respectively, and $i_c = \text{centerCol}$ and $j_c = \text{centerRow}$.

7.1 Detector Coordinates

We denote detector coordinates by (s, t). The center position of the pixel coordinates are given by the following equations:

$$\begin{aligned} s[i] &= \text{pixelWidth} * (i - \text{centerCol}), \text{ for } i = 0, \ldots, \text{numCols} - 1, \\ t[j] &= \text{pixelHeight} * (j - \text{centerRow}), \text{ for } j = 0, \ldots, \text{numRows} - 1, \end{aligned}$$

where pixelWidth and pixelHeight are assumed to be in length units, e.g., mm or cm. The indicies of the center detector column and center detector row are given by centerCol and centerRow, respectively. For a perfectly centered detector, centerCol $= 0.5(\text{numCols} - 1)$ and centerRow $= 0.5(\text{numRows} - 1)$.

7.2 Volume Parameterization

The location of the 3D voxels are specified by seven parameters as follows

$$\begin{aligned} x[i] &= \text{voxelWidth} * (i - 0.5 * (\text{numX} - 1)) + \text{offsetX}, \text{ for } i = 0, \ldots, \text{numX} - 1, \\ y[j] &= \text{voxelWidth} * (j - 0.5 * (\text{numY} - 1)) + \text{offsetY}, \text{ for } j = 0, \ldots, \text{numY} - 1, \\ z[k] &= \text{voxelHeight} * (k - 0.5 * (\text{numZ} - 1)) + \text{offsetZ}, \text{ for } k = 0, \ldots, \text{numZ} - 1, \end{aligned}$$

where voxelWidth, voxelHeight, offsetX, offsetY, and offsetZ are assumed to be in length units, e.g., mm or cm. Make sure that whatever units of length you choose that you stay consistent across all parameters. The units of the voxelized volume (i.e., the reconstruction volume) are

assumed to be in inverse length, e.g., mm^{-1} or cm^{-1}. We make this distinction because LEAP is fairly unique in it's quantitatively-accurate algorithms which we believe is very useful.

We note that there are some restrictions in the above volume parameters based on the CT geometry one is using. These restrictions are listed below.

- parallel-beam
  - voxelHeight must be equal to pixelHeight
  - offsetZ must be zero
  - numZ must be equal to numRows
- fan-beam
  - voxelHeight must be equal to pixelHeight
  - offsetZ must be zero
  - numZ must be equal to numRows
- cone-beam
  - no restrictions
- modular-beam
  - voxelHeight must be equal to voxelWidth

The restrictions for parallel- and fan-beam are that there is a one-to-one correspondence between the volume z-slices and the detector rows. If any of the above restrictions are violated, then LEAP will return an error when a forward projection, backprojection, or FBP algorithm is called.

Besides the above restrictions, there are recommended values for the voxel sizes. If the voxel size is significantly smaller or significantly bigger than these recommended sizes, then the LEAP algorithms may run much, much slower.

- parallel-beam
  - recommend that voxelWidth be equal to pixelWidth
- fan-beam
  - recommend that voxelWidth be equal to pixelWidth*sod/sdd
- cone-beam
  - recommend that voxelWidth be equal to pixelWidth*sod/sdd
  - recommend that voxelHeight be equal to pixelHeight*sod/sdd
- modular-beam
  - recommend that voxelWidth be equal to pixelWidth/2

It is recommended that one specify the volume parameters with the setDefaultVolume() command which sets the volume to fill the field of view of the CT system and use the recommended voxel sizes.

Note that the voxel samples are aligned with the x,y,and z axes and there are no methods to rotate these coordinates. This restriction is for a few reasons. First of all, there are certain

rotations that are incompatible with Separable Footprint projectors. Second, there are certain assumptions that one can leverage with standard CT geometry types (parallel, fan, and cone) that improve the computational efficiency of the projectors. In fact, the complexity of the CT geometry is highly related to the computation speed of it projectors. The speed of the projectors, listed from fasted to slowest, is: parallel-, fan-, cone-, and modular-beam. And finally, it is easier to subdivide the volume into smaller chunk to process operations across multiple GPUs or to split it up into chunks that can fit into the GPU memory. All of that being said, there are two *tricks* to perform some rotations of the voxel coordinates. First, for modular-beam coordinates, one may simply perform the inverse rotation on the location and orientation of the source locations and detector locations and orientations that will effectively rotate the volume coordinates. Second, one may provide a phase shift of the provided projection angles which effectively rotates the volume coordinates around the z-axis. Lastly, one may utilize the offsetX, offsetY, and offsetZ parameters to perform projections and backprojections of a single 2D plane or a single voxel at a time to generate a custom voxel grid. This scheme, however, is less efficient and more difficult to manage.

In the next section we describe the integral transforms and inversion algorithms for parallel-, fan-, and cone-beam geometries.

7.3 X-ray Transform and Its Adjoint

Consider a linear operator denoted by $\mathcal{A}$. Then its adjoint, usually denoted by $\mathcal{A}^*$, is defined such that

$$< \mathcal{A}x, y >=< x, \mathcal{A}^* y > .$$

Note that if $\mathcal{A}$ where a real-valued matrix, its adjoint would be equal to its transpose.

The X-ray Transform of $f \in L^1(\mathbb{R}^3)$ is a linear operator given by

$$Pf(\mathbf{y}, \Theta) = \int_{\mathbb{R}} f(\mathbf{y} + l\Theta)\, dl,$$

where in general $y, \Theta \in \mathbb{R}^3$, but the exact specification of these parameters depends on the geometry of the CT system. The X-ray Transform is often referred to as a projection or *forward projection* of a function. While the adjoint of the X-ray Transform is commonly referred to as the *backprojection* operator. The backprojection operator usually takes the form of an integral over the various measured angles of the X-ray transform, but its exact form depends on the parameterization of the line integrals in the forward projection.

In the following, we shall make use of the following definitions

$$\begin{aligned} \theta &:= \theta(\varphi) := \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ 0 \end{bmatrix} \\ \theta^\perp &:= \theta^\perp(\varphi) := \begin{bmatrix} -\sin\varphi \\ \cos\varphi \\ 0 \end{bmatrix}. \end{aligned}$$

We shall usually denote $g = Pf$.

7.4 Parallel-Beam

The X-ray transform and its adjoint in parallel-beam coordinates are given by

$$\begin{aligned} Pf(s, \varphi, x_3) &:= \int_{\mathbb{R}} f(s\boldsymbol{\theta}^\perp(\varphi) - l\boldsymbol{\theta}(\varphi) + x_3\mathbf{e}_3)\, dl \\ P^* g(\boldsymbol{x}) &= \int_0^{2\pi} g(\boldsymbol{x} \cdot \boldsymbol{\theta}^\perp(\varphi), \varphi, x_3)\, d\varphi \end{aligned}$$

Note that $g(s,\varphi) = g(-s,\varphi \pm \pi)$; the rays defined by these two coordinates are known are conjugate rays.

The object, f, can be reconstructed by

$$f(\boldsymbol{x}) \quad = \quad \frac{1}{4\pi}\int_0^{2\pi}\int_{\mathbb{R}} h(\boldsymbol{x}\cdot\boldsymbol{\theta}^{\perp}(\varphi)-s)g(s,\varphi,x_3)\,ds\,d\varphi.$$

The discretization of the above equation is referred to as filtered backprojection (FBP). The function h is referred to as the ramp filter and is described in Section 10.1.

7.5 Fan-Beam

The X-ray transform and its adjoint in fan-beam coordinates (with linear detector) with center of rotation offset of τ, source to object distance (sod) R, and source to detector distance D, are given by

$$\begin{aligned} Pf(u,\beta,x_3) &:= \int_{\mathbb{R}} f\left(R\boldsymbol{\theta}(\beta)-\tau\boldsymbol{\theta}^{\perp}(\beta)-\frac{l}{\sqrt{1+u^2}}\left[\boldsymbol{\theta}(\beta)-u\boldsymbol{\theta}^{\perp}(\beta)\right]+x_3\mathbf{e}_3\right)\,dl \\ P^*g(\boldsymbol{x}) &= \int \frac{1}{R-\boldsymbol{x}\cdot\boldsymbol{\theta}(\beta)}\sqrt{1+u^2(\boldsymbol{x},\beta)}\,g\left(u(\boldsymbol{x},\beta),\beta,x_3\right)\,d\beta \\ u(\boldsymbol{x},\beta) &:= \frac{\boldsymbol{x}\cdot\boldsymbol{\theta}^{\perp}(\beta)+\tau}{R-\boldsymbol{x}\cdot\boldsymbol{\theta}(\beta)} \end{aligned}$$

The parameter u is the detector column coordinate divided by sdd and can be viewed as the slope of the ray in the xy plane. Note the conjugate ray relation is

$$g(u,\beta,x_3) = g\left(\frac{-u+\frac{2\tau R}{R^2-\tau^2}}{1+u\left(\frac{2\tau R}{R^2-\tau^2}\right)},\beta-2\tan^{-1}u+\tan^{-1}\left(\frac{2\tau R}{R^2-\tau^2}\right)\pm\pi,x_3\right).$$

The object, f, can be reconstructed by the following Filtered Backprojection (FBP) algorithm

$$f(\boldsymbol{x}) \quad = \quad \frac{R}{2\pi}\int\frac{1}{(R-\boldsymbol{x}\cdot\boldsymbol{\theta}(\beta))^2}\int_{\mathbb{R}} h\left(u(\boldsymbol{x},\beta)-u\right)g(u,\beta,x_3)m(\tan^{-1}u,\beta)\frac{1+\frac{\tau}{R}u}{\sqrt{1+u^2}}\,du\,d\beta.$$

The function m accounts for the number of redundant measurements for a particular angle and for an angular range of less than 360° it is often referred to as the *Parker Weights*; See Section 7.7. The function h is referred to as the ramp filter and is described in Section 10.1.

Parallel-beam and fan-beam coordinates are related by the following transformations:

$$\begin{aligned} s &= \frac{Ru-\tau}{\sqrt{1+u^2}}, \\ \varphi &= \beta-\tan^{-1}u \end{aligned}$$

7.6 Cone-Beam

The X-ray transform and its adjoint in cone-beam coordinates (with planar detector) with center of rotation offset of τ and source trajectory radius R are given by

$$\begin{aligned} Pf(u,\beta,v) &:= \int_{\mathbb{R}} f\left(R\boldsymbol{\theta}(\beta) - \tau\boldsymbol{\theta}^{\perp}(\beta) + \frac{l}{\sqrt{1+u^2+v^2}}\left[-\boldsymbol{\theta}(\beta) + u\boldsymbol{\theta}^{\perp}(\beta) + v\widehat{\boldsymbol{z}}\right]\right) dl \\ P^*g(\boldsymbol{x}) &= \int \frac{\sqrt{1+u^2(\boldsymbol{x},\beta)+v^2(\boldsymbol{x},\beta)}}{(R-\boldsymbol{x}\cdot\boldsymbol{\theta}(\beta))^2} g\left(u(\boldsymbol{x},\beta),\beta,v(\boldsymbol{x},\beta)\right)\, d\beta \\ u(\boldsymbol{x},\beta) &:= \frac{\boldsymbol{x}\cdot\boldsymbol{\theta}^{\perp}(\beta)+\tau}{R-\boldsymbol{x}\cdot\boldsymbol{\theta}(\beta)} \\ v(\boldsymbol{x},\beta) &:= \frac{x_3}{R-\boldsymbol{x}\cdot\boldsymbol{\theta}(\beta)} \end{aligned}$$

The parameter v is the detector row coordinate divided by sdd and can be viewed as the slope of the ray.

The object, f, can be (approximately) reconstructed by the following Filtered Backprojection (FBP) algorithm

$$\begin{aligned} f(\boldsymbol{x}) &\approx \frac{R}{2\pi}\int \frac{1}{(R-\boldsymbol{x}\cdot\boldsymbol{\theta}(\beta))^2} \\ &\times \int_{\mathbb{R}} h(u(\boldsymbol{x},\beta)-u) g(u,\beta,v(\boldsymbol{x},\beta)) m(\tan^{-1}u,\beta) \frac{1+\frac{\tau}{R}u}{\sqrt{1+u^2+v^2(\boldsymbol{x},\beta)}}\, du\, d\beta. \end{aligned}$$

The function h is referred to as the ramp filter and is described in Section 10.1.

We would like to take a special note in the exact form of the adjoint of the cone-beam X-ray Transform. Notice that

$$P^*g(\mathbf{x}) \neq \int_{\mathbb{R}} g(u(\mathbf{x},\beta),\beta,v(\mathbf{x},\beta))\, d\beta.$$

The right hand side of the above equation is what many say is the backprojection operator, but this is false. A backprojection is not always merely an integral over all the measured angles for those rays going through a specific point in space. Thus it is also often said that FBP reconstruction of cone-beam data requires a *weighted* backprojection because of the $\frac{1}{R-\mathbf{x}\cdot\theta}$ term, but we see that this term is actually present in the backprojection already. This origin of this term is the Jacobian of the change of coordinates matrix when deriving the backprojection. One of the nice things about using match projector pairs, is that this weighting term is implicitly included when performing the backprojection operation.

7.7 Parker Weighting

Define $\alpha := \tan^{-1}(u)$. The function $m(\cdot,\cdot)$ given above is known as the Parker weighting function. For a short scan, where $\pi + 2\alpha_{max} + \tan^{-1}\left(\frac{2\tau R}{R^2-\tau^2}\right) \leq \beta_{end} - \beta_0 < 2\pi$, it is given by

$$m(\alpha,\beta) := \begin{cases} \sin^2\left(\frac{\pi}{4}\frac{\beta-\beta_0}{\alpha_{thres}-\alpha}\right), & 0 \leq \beta-\beta_0 < 2(\alpha_{thres}-\alpha), \\ 1, & 2(\alpha_{thres}-\alpha) \leq \beta-\beta_0 < \pi - 2\alpha - \tan^{-1}\left(\frac{2\tau R}{R^2-\tau^2}\right), \\ \cos^2\left(\frac{\pi}{4}\frac{(\beta-\beta_0)+2\alpha+\tan^{-1}\left(\frac{2\tau R}{R^2-\tau^2}\right)-\pi}{\alpha_{thres}+\alpha}\right), & \pi - 2\alpha - \tan^{-1}\left(\frac{2\tau R}{R^2-\tau^2}\right) \leq \beta-\beta_0 < \pi + 2\alpha_{thres}, \end{cases}$$

where $\alpha_{thres} = \frac{\beta_{end}-\beta_0-\pi}{2}$. For a full scan, i.e., when $\beta_{end}-\beta_0 = 2\pi$, we have $m(\alpha,\beta) = \frac{1}{2}$.

7.8 Modular Beam

The modular beam data type provides a flexible geometry format to specify systems that do not fall into one of the above coordinate systems. Since the data is not necessarily given on a regular grid, one can only reconstruction these data sets with iterative techniques. To use this type of geometry one must specify the location of every source and detector module pair along with the detector module orientation. A sketch of an example system modeled by the modular beam geometry is shown in Figure 4.

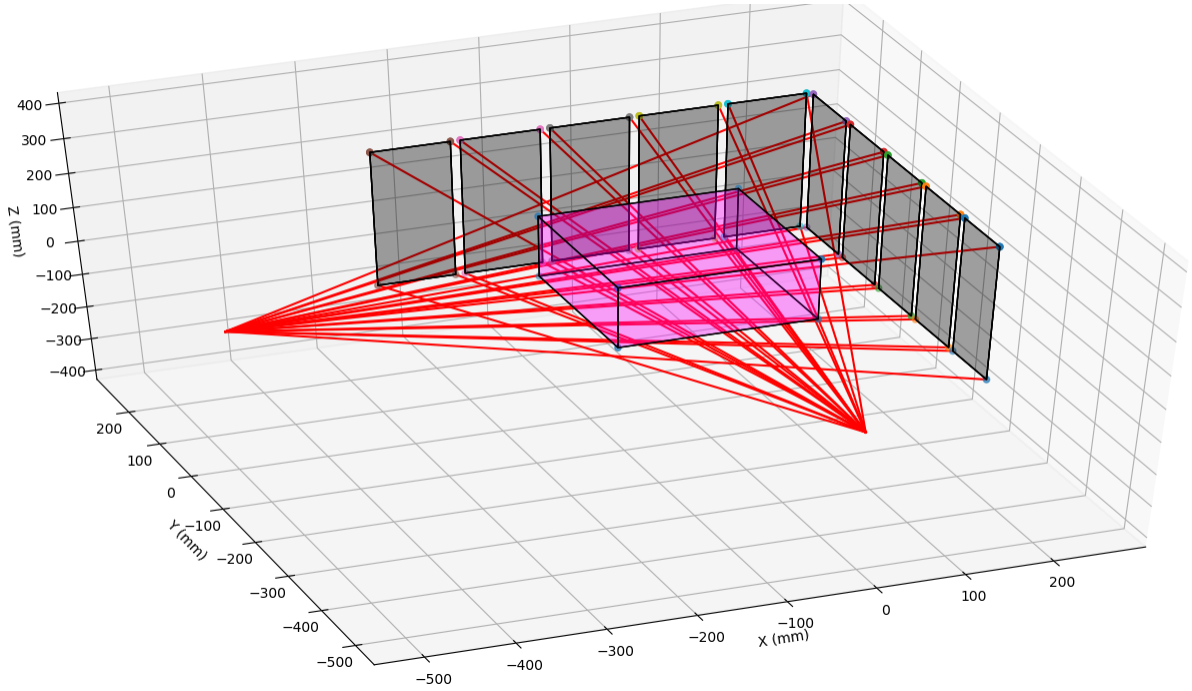

Figure 4: Sketch of a geometry built with the modular-beam framework. The modules here lie on the perimeter of two sides of a rectangle as might exist for a carry-on luggage x-ray scanner. This sketch was generated with the *sketchSystem()* Python function. The magenta box is the reconstruction volume, the gray rectangles are the detector panels, and the red lines trace from the source to each of the four corners of the detector.

8 Attenuated Radon Transform

The Attenuated Radon Transform (ART) and its adjoint in parallel-beam coordinates are given by

$$\begin{aligned} P_\mu f(s,\varphi,x_3) &= \int_{\mathbb{R}} f(s\boldsymbol{\theta}^\perp(\varphi) - l\boldsymbol{\theta}(\varphi) + x_3\mathbf{e}_3) e^{-\int_0^\infty \mu(s\boldsymbol{\theta}^\perp(\varphi)-(l+m)\boldsymbol{\theta}(\varphi))\,dm}\,dl \\ P_\mu^* g(\boldsymbol{x}) &= \int_0^{2\pi} g(\boldsymbol{x}\cdot\boldsymbol{\theta}^\perp(\varphi),\varphi,x_3) e^{-\int_0^\infty \mu(\mathbf{x}-m\boldsymbol{\theta}(\varphi))\,dm}\,d\varphi \end{aligned}$$

The object, f, can be reconstructed by the following Filtered Backprojection (FBP) algorithm

$$f(\boldsymbol{x}) = \frac{1}{4\pi}\text{Re div}\int_0^{2\pi} \boldsymbol{\theta}^\perp(\varphi) e^{\int_0^\infty \mu(\mathbf{x}-m\boldsymbol{\theta}(\varphi))\,dm} \left(e^{-h} H e^{h} g\right)(\boldsymbol{x}\cdot\boldsymbol{\theta}^\perp(\varphi),\varphi,x_3)\,d\varphi,$$

where $h := \frac{1}{2}(I + iH)P\mu$, I is the identify operator, and H is the Hilbert Transform.

9 Projectors for Cylindrically-Symmetric Objects

The X-ray Transform for cylindrically-symmetric objects is the same as those X-ray Transforms for standard objects, but the volume is assumed to be cylindrically antisymmetric (the left and right sides can be different). This object model causes the adjoint transform to take a very different form where there is a singularity along the axis of symmetry. The FBP reconstruction algorithms are the same as those stated in the previous sections. This is achieved by leveraging the fact that the object is symmetric and thus projections from all angles are the same.

10 Digital Filters for CT

Typical digital signal processing (DSP) texts are not motivated by imaging applications. Many filters in these texts are designed for the one-dimensional DSP, such as the audio processing field and are ill-suited for imaging applications. For example, many DSP texts state the superior interpolation performance of low-pass filters that approximate the *ideal* low-pass filter (a rect function) over linear interpolation. These approximations to the *ideal* low-pass filter have sharp transitions in their frequency response which result in oscillatory impulse responses. If used in imaging applications, these filters cause ringing artifacts (Gibb's phenomena) in the image and may also result in negative values in the filtered signal which may also be undesirable. The sinc interpolation kernel of an *ideal* low-pass filter is, in fact, just as arbitrary as any other interpolation kernel. In some applications, a polynomial interpolation kernel will produce a more desired result. Linear interpolation is computationally efficient, does not cause ringing artifacts, does not boost noise, produces a nonnegative output (provided that the input is nonnegative), is flexible, and easy to implement.

In this section we derive a collection of digital filters specifically designed for use in tomographic image reconstruction. This includes the following filters: ramp, derivative, Hilbert, and low-pass. We start with the ramp filter.

10.1 Ramp Filter

This section is focused on deriving discrete impulse responses for the ramp filter. The (continous space) ramp filter for $f \in L^1(\mathbb{R}^n)$ is given by

$$\mathcal{R}f(x) := c_n \int_{\mathbb{R}^n} \frac{f(y)}{\|x-y\|^{n+1}}\, dy, \tag{1}$$

$$c_n := -\frac{2}{(2\sqrt{\pi})^n}\frac{\Gamma(n)}{\Gamma(n/2)}, \tag{2}$$

where $c_n = -\frac{1}{\pi}, -\frac{1}{2\pi}, -\frac{1}{\pi^2}$ for $n = 1, 2, 3$, respectively. We also define the Fourier transform by

$$\mathcal{F}f(\xi) := F(\xi) := \int_{\mathbb{R}^n} f(x) e^{-2\pi i <x,\xi>}\, dx. \tag{3}$$

Then $\mathcal{F}\{\mathcal{R}f\}(\xi) = \|2\pi\xi\| F(\xi)$.

We define the Hilbert transform for $f \in L^1(\mathbb{R})$ by

$$\mathcal{H}f(s) := \frac{1}{\pi}\int_{\mathbb{R}} \frac{f(t)}{s-t}\, dt. \tag{4}$$

With respect to Fourier transforms, the Hilbert transform is given by

$$\mathcal{F}\{\mathcal{H}f\}(\sigma) = i\text{sgn}(\sigma) F(\sigma).$$

If we let $\mathcal{D} = \frac{d}{ds}$ be the derivative operator, then $\mathcal{F}\{\mathcal{D}f\}(\sigma) = -2\pi i\sigma F(\sigma)$. Therefore we have for $n = 1$ that $\mathcal{R} = \mathcal{D}\mathcal{H} = \mathcal{H}\mathcal{D}$. We will exploit this property in the derivation of our discrete ramp filters.

Ramp filters are used in both analytic and iterative Computed Tomography (CT) image reconstruction algorithms. The most common analytic CT image reconstruction algorithms are the filtered backprojection (FBP) and backprojection filtration (BPF) algorithms which require one and two dimensional ramp filters, respectively. In iterative reconstruction one can use the 2D ramp filter as a preconditioner for the gradient descent or conjugate gradient algorithms to improve their rate of convergence.

The topic of digital ramp filter design has been addressed in a large amount of papers. Most of these papers derive the digital impulse response by windowing the ideal ramp filter in frequency space. Although this method of filter design allows the user to exactly specify the frequency response, the impulse response of many of these filters are highly oscillatory and thus produce results that are highly oscillatory which is undesirable. For speed, application of the ramp filter is performed in the frequency domain using fast Fourier transform (FFT) operations, but all ramp filters are defined in the spatial domain to avoid a negative bias on the result [**?**].

10.1.1 Derivation of Ramp Filters

The band-limited filter with least L^2 error is given by

$$\begin{aligned} h_{ramp}(s) &= \int_{-1/2}^{1/2} 2\pi|\sigma|e^{2\pi is\sigma}\,d\sigma \\ &= 4\pi\int_{0}^{1/2} \sigma\cos(2\pi s\sigma)\,d\sigma \\ &= \frac{\pi s\sin(\pi s)+\cos(\pi s)-1}{\pi s^2} \end{aligned}$$

and the discrete filter is given by $h_{ramp}[k] = h_{ramp}(k) = \frac{(-1)^k-1}{\pi k^2}$. This is known as the Ram-Lak filter and thus we define $h_{RL}[k] := h_{ramp}[k]$. Although the frequency response of this filter is ideal, the impulse response oscillates with every sample as shown in Figure 6. Thus we see that convolution with this filter will invariably produce oscillatory functions which is undesired. To understand and mitigate this effect we focus on the Hilbert transform which is one part of the ramp filter.

Now the band-limited Hilbert filter is given by

$$\begin{aligned} h_{hilb}(s) &:= \int_{-1/2}^{1/2} i\mathrm{sgn}(\sigma)e^{2\pi is\sigma}\,d\sigma &(5)\\ &= \frac{1-\cos(\pi s)}{\pi s}. &(6)\end{aligned}$$

Note that $h'_{hilb}(s) = h_{ramp}(s)$, as expected. Now consider two discrete Hilbert filters

$$\begin{aligned} h_{hilb}[k] &:= h_{hilb}(k) = \frac{1-(-1)^k}{\pi k} &(7)\\ h_{hilb,1/2}[k] &:= h_{hilb}(k-1/2) = \frac{1}{\pi(k-\frac{1}{2})}. &(8)\end{aligned}$$

In Figure 5 we notice that while $h_{hilb}[k]$ is oscillatory, $h_{hilb,1/2}[k]$ is not. This oscillatory behavior can thus be removed by introduceing a (backward) half sample shift into the filter. Now we can define a ramp filter by convolving $h_{hilb,1/2}[k]$ with a finite difference filter with a half sample forward shift.

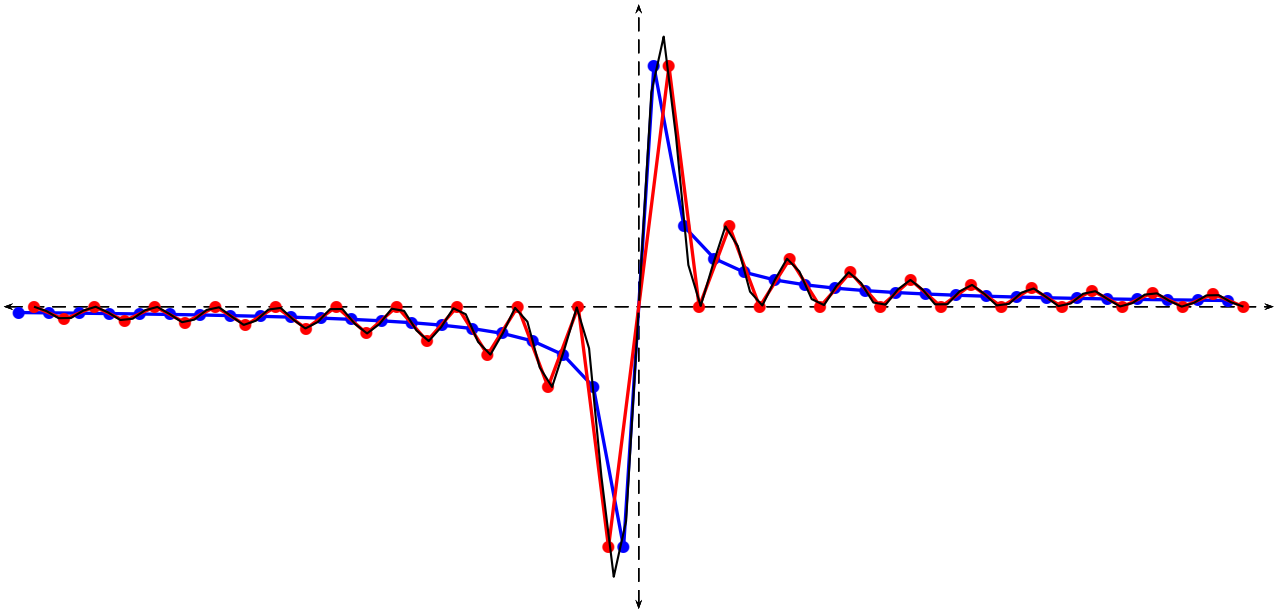

Figure 5: Discrete impulse response of the ideal Hilbert filter with zero shift (red) and half shift (blue). The continuous space impulse response is shown in black.

We find the finite difference coefficients by solving the following linear system

$$\begin{bmatrix} 2 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 & \cdots & 2M-1 \\ 1 & 3^3 & 5^3 & \cdots & (2M-1)^3 \\ 1 & 3^5 & 5^5 & \cdots & (2M-1)^5 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 3^{2M-1} & 5^{2M-1} & \cdots & (2M-1)^{2M-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{M-1} \end{bmatrix} \tag{9}$$

$$h_{d,2M}[k] := \frac{1}{2} \begin{cases} -a_{-k}, & k = -M+1, -M+2, \ldots 0, \\ a_{k-1}, & k = 1, 2, \ldots, M \end{cases} \tag{10}$$

and thus for $M = 1, 2, 3, 4, 5$ the filter coefficients are given by

$$\{h_{d,2}[k]\}_{k=0}^{1} := \{-1, 1\} \tag{11}$$

$$\{h_{d,4}[k]\}_{k=-1}^{2} := \left\{\frac{1}{24}, -\frac{9}{8}, \frac{9}{8}, -\frac{1}{24}\right\} \tag{12}$$

$$\{h_{d,6}[k]\}_{k=-2}^{3} := \left\{-\frac{3}{640}, \frac{25}{384}, -\frac{75}{64}, \frac{75}{64}, -\frac{25}{384}, \frac{3}{640}\right\} \tag{13}$$

$$\{h_{d,8}[k]\}_{k=-3}^{4} := \left\{\frac{5}{7168}, -\frac{49}{5120}, \frac{245}{3072}, -\frac{1225}{1024}, \frac{1225}{1024}, -\frac{245}{3072}, \frac{49}{5120}, -\frac{5}{7168}\right\} \tag{14}$$

$$\{h_{d,10}[k]\}_{k=-4}^{5} := \left\{-\frac{35}{294912}, \frac{405}{229376}, -\frac{567}{40960}, \frac{735}{8192}, -\frac{19845}{16384}, \frac{19845}{16384}, -\frac{735}{8192}, \frac{567}{40960}, -\frac{405}{229376}, \frac{35}{294912}\right\} \tag{15}$$

Note that $h_{d,M}$ has M nonzero entries and is accurate to the Mth order, i.e., $f'\left(\frac{1}{2}T\right) = f(kT) * h_{d,M}[k] + O(T^M)$ as $T \to 0$. The impulse response of the limiting case is given by

$$h_{d,\infty}[k] := \lim_{M\to\infty} h_{d,M}[k] = \frac{(-1)^k}{\pi\left(k+\frac{1}{2}\right)^2}. \tag{16}$$

The impulse response of the M-th order digital ramp filter is given by

$$h_M[k] := h_{hilb,1/2} * h_{d,M}[k], \tag{17}$$

where

$$\begin{aligned}
h_2[k] &:= \frac{1}{\pi\left(\frac{1}{4}-k^2\right)} && (18)\\
h_4[k] &:= \frac{1}{\pi\left(\frac{1}{4}-k^2\right)}\frac{k^2-\frac{5}{2}}{k^2-\frac{9}{4}} && (19)\\
h_6[k] &:= \frac{1}{\pi\left(\frac{1}{4}-k^2\right)}\frac{k^4-\frac{35}{4}k^2+\frac{259}{16}}{\left(k^2-\frac{9}{4}\right)\left(k^2-\frac{25}{4}\right)} && (20)\\
h_8[k] &:= \frac{1}{\pi\left(\frac{1}{4}-k^2\right)}\frac{k^6-\frac{336}{16}k^4+\frac{1974}{16}k^2-\frac{3229}{16}}{\left(k^2-\frac{9}{4}\right)\left(k^2-\frac{25}{4}\right)\left(k^2-\frac{49}{4}\right)} && (21)\\
h_{10}[k] &:= \frac{1}{\pi\left(\frac{1}{4}-k^2\right)}\frac{k^8-\frac{165}{4}k^6+\frac{4389}{8}k^4-\frac{86405}{32}k^2+\frac{1057221}{256}}{\left(k^2-\frac{9}{4}\right)\left(k^2-\frac{25}{4}\right)\left(k^2-\frac{49}{4}\right)\left(k^2-\frac{81}{4}\right)} && (22)
\end{aligned}$$

and the corresponding frequency responses are given by

$$\begin{aligned}
H_2(X) &:= [2\sin(\pi X)]\,\mathrm{sgn}(X) && (23)\\
H_4(X) &:= \left[\frac{9}{4}\sin(\pi X)-\frac{1}{12}\sin(3\pi X)\right]\mathrm{sgn}(X) && (24)\\
H_6(X) &:= \left[\frac{75}{32}\sin(\pi X)-\frac{25}{192}\sin(3\pi X)+\frac{3}{320}\sin(5\pi X)\right]\mathrm{sgn}(X) && (25)\\
H_8(X) &:= \left[\frac{1225}{512}\sin(\pi X)-\frac{245}{1536}\sin(3\pi X)\right.\\
&+ \left.\frac{49}{2560}\sin(5\pi X)-\frac{5}{3584}\sin(7\pi X)\right]\mathrm{sgn}(X) && (26)\\
H_{10}(X) &:= \left[\frac{19845}{8192}\sin(\pi X)-\frac{735}{4096}\sin(3\pi X)+\frac{567}{20480}\sin(5\pi X)\right.\\
&- \left.\frac{405}{114688}\sin(7\pi X)+\frac{35}{147456}\sin(9\pi X)\right]\mathrm{sgn}(X) && (27)
\end{aligned}$$

for $X \in \left[-\frac{1}{2},\frac{1}{2}\right)$. The impulse and frequency responses of these filters are shown in Figures 6 and 7, respectively. Note that $h_\infty[k] = h_{RL}[k]$.

We additionally define $h_0[k] := h_2[k] * \left\{\frac{1}{4},\frac{1}{2},\frac{1}{4}\right\} = h_2[k]\frac{k^2-\frac{3}{4}}{k^2-\frac{9}{4}}$ and $H_0(X) = H_2(X)\frac{1+\cos(2\pi X)}{2}$. The zero subscript here does not specify the order of the finite difference as in the other filter definitions, but was chosen to uniformity of notation and to denote that the frequency response of this filter at Nyquist is zero.

Note that the sign of the derivative of these filters only changes three times, while the sign of the derivative of $h_{RL}[n]$ changes with every sample.

Since $h_M[k]$ have infinite impulse response (IIR), one must window these filters. Suppose one wishes to filter $g[k]$, where $g[k] \neq 0$ for $k = -N/2,\ldots,N/2-1$ with the ramp filter. Then one can filter the data using FFT operations by

$$IFFT_{2N}(FFT_{2N}(g)FFT_{2N}(wh_M))[k],$$

where $w[k]$ is a window function ($w[k] = 0$ for $k \notin [-N, N-1]$) and FFT_{2N} and $IFFT_{2N}$ are the $2N$ point Fast Fourier Transform and Inverse Fast Fourier Transform operations, respectively.

Since $h_M[k] = O(k^{-2})$, the ramp filter decays rapidly and thus one can use the rectangular window given by

$$w[k] = \begin{cases} 1, & k = -N, -N+1, \ldots, N-1, \\ 0, & \text{otherwise} \end{cases}$$

without any significant distortion of the frequency response. It is advised that one performs the filtering in this fashion, rather than using equations (23-27) explicitly.

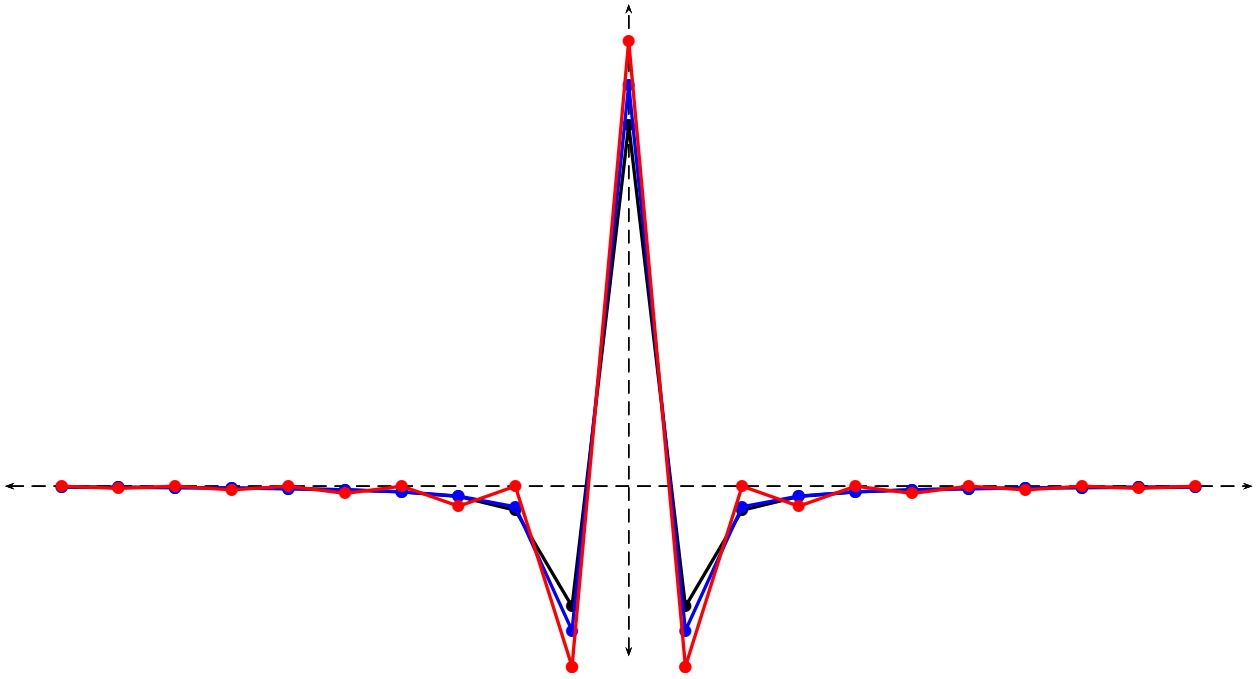

Figure 6: Impulse response of Ram-Lak (red), $h_4[k]$ (blue), and $h_2[k]$ (Shepp-Logan, black) ramp filters.

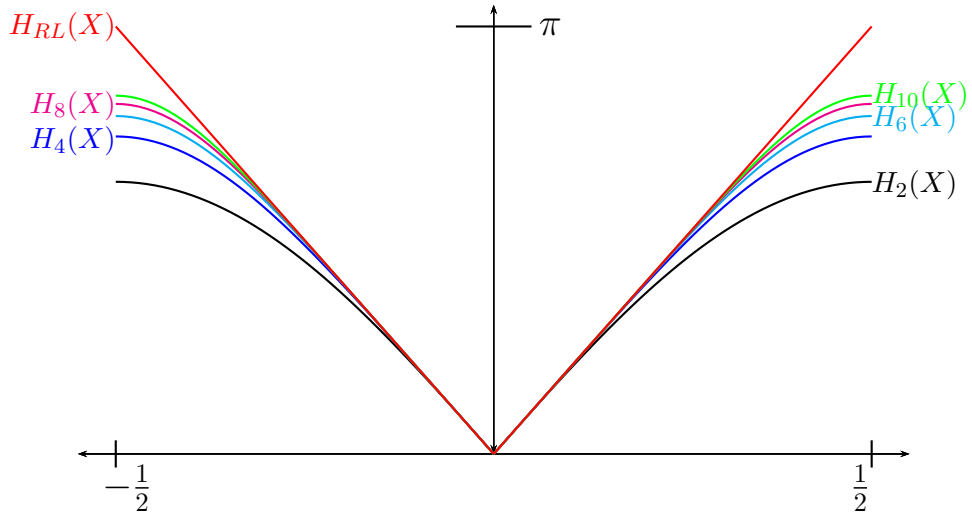

Figure 7: Frequency response of Ram-Lak (red), $H_{10}(X)$ (green), $H_8(X)$ (magenta), $H_6(X)$ (cyan), $H_4(X)$ (blue), and $H_2(X)$ (Shepp-Logan, black) ramp filters.

Higher order filters can be found using equations (8, 9, 10, 17), but are unlikely to provide more accurate reconstructions because they will further amplify noise and ringing artifacts and some of the gained resolution will be lost in the interpolation methods used in the backproject

Table 4: Relative L^2 difference $\left(\frac{\|H_M - H_{RL}\|}{\|H_{RL}\|}\right)$ between $H_M(X)$ and $H_{RL}(X)$.

Filter	Relative L^2 Difference
$H_2(X)$	24.5%
$H_4(X)$	14.7%
$H_6(X)$	10.9%
$H_8(X)$	8.7%
$H_{10}(X)$	7.4%

step of FBP or BPF.

10.1.2 Two Dimensional Extensions

The filters defined above can be extended to two dimensions by the following

$$H_{2D}(X,Y) \quad := \quad \sqrt{H^2(X) + H^2(Y) - \frac{1}{H^2(1/2)} H^2(X) H^2(Y)}. \tag{28}$$

Note that $H_{2D}(X,0) = H(X)$, $H_{2D}(0,Y) = H(Y)$, and $H_{2D}(X,\pm 1/2) = H_{2D}(\pm 1/2, Y) = H(1/2)$. Thus we see that $H_{2D}(X,Y)$ (including its periodic extension) is infinitely differentiable everywhere except at the origin. This ensures that the impulse response will decay rapidly.

One can show and it is well known that $\mathcal{R} = -\Delta$, where $\Delta = \sum_i \frac{\partial^2}{\partial x_i^2}$ is the Laplace operator. Using $H(X) = H_2(X)$, we have that

$$H_{2D}^2(X,Y) = 4\sin^2(\pi X) + 4\sin^2(\pi Y) - 4\sin^2(\pi X)\sin^2(\pi Y)$$

which is the frequency response of

$$-\begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & -3 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix}.$$

This is a common filter for the discrete Laplacian, where the weight of the derivative on the diagonals are weighted by their distance from the center sample. Thus we see that our choice of $H_{2D}(X,Y)$ is not only practical, but also theoretically relevant.

It is sometimes advantageous to add a lowpass filter, $U(X)$, to the ramp filter by $H_{LP}(X) := U(X)H(X)$, where $U(0) = 1$ and $U(\pm 1/2) = 0$. One can extend this to the 2D ramp filter by

$$H_{2D,LP}(X,Y) \quad := \quad U(X)U(Y)\sqrt{H^2(X) + H^2(Y) - \frac{1}{H^2(1/2)} H^2(X) H^2(Y)}. \tag{29}$$

10.1.3 Edge Response and Modular Transfer Functions

We illustrate the resolution of the filters defined above by performing a simulation experiment. We simulated a circular disk and reconstructed it with a collection of filters. The edge spread function (ESF) and modular transfer function (MTF) plots are shown in Figure 8. We also include a common ramp filter, which we denote $h_{Butterworth}[k]$ and is given by $H_{Butterworth}(X) = H_{RL}(X)\frac{1}{1+X^{10}}$. Note that the ESF of $h_{Butterworth}[k]$ is highly oscillatory (even more so than the Ram-Lak filter). The filter $h_4[k]$ has nearly identical resolution as $h_{Butterworth}[k]$, but with very little ringing.

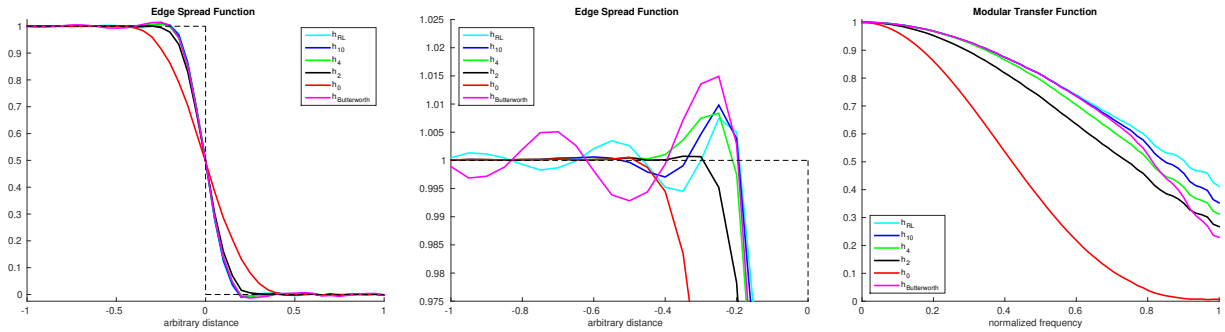

Figure 8: The ESF and MTF using $h_0[k]$, $h_2[k]$, $h_4[k]$, $h_{10}[k]$, $h_{RL}[k]$, and $h_{Butterworth}[k]$.