Master thesis report

Divergent X-ray tomography reconstruction and optimisation

École Polytechnique Fédérale de Lausanne
Rte Cantonale, 1015 Lausanne, Switzerland

Supervisors :
Sepand KASHANI | sepand.kashani@epfl.ch
Dr. Matthieu SIMEONI | matthieu.simeoni@epfl.ch
Dr. Edward ANDÒ | edward.ando@epfl.ch

May - November 2023

Author :
Youssef HAOUCHAT | youssef.haouchat@ensta-paris.fr

Master student of ENSTA Paris, France
École Nationale Supérieure des Techniques Avancées

Parcours : ModSim

ENSTA Paris supervisor :
Laure GIOVANGIGLI | laure.giovangigli@ensta-paris.fr
Associate Professor of Mathematics at ENSTA Paris

Contents

List of Figures

Acknowledgements

Firstly, I would like to express my deepest gratitude to Mr. Sepand KASHANI, a dedicated PhD student at EPFL's LCAV laboratory with whom I have worked a lot with. He found a very pleasant balance between autonomy and regular supervision, and was able to guide me through the theoretical and computational aspects of my project. Always with great sympathy, he was able to lead my curiosity through new scientific points of view. Finally, huge thanks for the excellent proofreading of this report.

I would also like to convey my sincerest appreciation to Mr. Matthieu SIMEONI and Mr. Edward ANDÒ for having judged me capable of maintaining this project of the Center For Imaging and for having brought an expert scientific point of view, both theoretical and experimental, and the necessary hindsight to the smooth running of my master thesis.

In extending my heartfelt gratitude, I cannot overlook the warm and enthusiastic welcome I received from every member of the laboratory. Each one of you played an integral role in making me feel not just welcome, but a part of this esteemed group. I must express my gratitude to my supervisors for having reviewed this report and having given constructive feedback.

I would also like to acknowledge Mr. Michael UNSER, head of the Biomedical Imaging Group for introducing me to some of his work and projects, and for trusting me by opening the doors of his laboratory for the next stage of my professional project.

Notations

Sets and Functional Spaces

$\mathbb{Z}$	Integers	
$\mathbb{R}$	Real numbers	
$\mathbb{C}$	Complex numbers	
$\mathbb{Z}^n$	Vectors of integers of dimension n	
$\mathbb{R}^n$	Vectors of real numbers of dimension n	
$\mathbb{S}^{n-1}$	Unit sphere embedded in $\mathbb{R}^n$	$\mathbb{S}^{n-1} = \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\|_2 = 1\}$
$L_1(\mathbb{R}^n)$	Absolutely-integrable measurable functions in $\mathbb{R}^n$	
$L_2(\mathbb{R}^n)$	Finite-energy functions in $\mathbb{R}^n$	
$\mathcal{H}/\mathcal{H}^*$	Hilbert space/ its dual	

X-ray tomography Model

$f, f(\boldsymbol{x})$	Volumetric object	$\mathbb{R}^n \to \mathbb{R}$
φ	Basis function	$\mathbb{R}^n \to \mathbb{R}$
$\boldsymbol{\theta}$	Unitary projection direction	$\boldsymbol{\theta} \in \mathbb{S}^{n-1}$
$\boldsymbol{\theta}^\perp$	Hyperplane orthogonal to $\boldsymbol{\theta}$	$\text{span}(\boldsymbol{\theta}) \bigoplus \boldsymbol{\theta}^\perp = \mathbb{R}^n$
$\boldsymbol{p}$	Offset, in the hyperplane $\boldsymbol{\theta}^\perp$	$\boldsymbol{p} \in \mathbb{R}^n$
$X_{\boldsymbol{\theta}}[\cdot]$	X-ray transform (3D geometry given by $\boldsymbol{\theta}$)	$L_2(\mathbb{R}^3) \to L_2(\mathbb{R}^2)$
$X^*_{\boldsymbol{\theta}}[\cdot]$	Adjoint operator of $X_{\boldsymbol{\theta}}$ (backprojection)	$L_2(\mathbb{R}^2) \to L_2(\mathbb{R}^3)$
N	Number of voxels of the volumetric object	$\mathbb{N}^*$
c, c_k	Discrete coefficients of f	$\mathbb{R}^N$
$(h_i)_{i \in [\![1 \dots M]\!]}$	Sampling linear forms of the operator	$h_i \in \mathcal{H}^*$
M	Number of X-ray acquisitions (rays)	$\mathbb{N}^*$
y	Measurement dataset	$\mathbb{R}^M$
A/A^T	Matrix/Transpose of the discrete operator	$\mathbb{R}^{M \times N}$
$\lambda(A)$	Eigenvalue of operator A	$\mathbb{C}$
$\boldsymbol{n}$	Additive random noise	$\mathbb{R}^M$

Reconstruction

$\hat{c}$	Estimate of c	$\mathbb{R}^N$
$\mathcal{F}$	Data fidelity term of the objective functional	$\mathbb{R}^N \to \mathbb{R}$
$\mathcal{R}$	Convex regularisation functional	$\mathbb{R}^N \to \mathbb{R}$
η	Regularisation parameter	$\mathbb{R}_+$
$\text{prox}_{\mathcal{R}}$	Proximal operator of $\mathcal{R}$	$\mathbb{R}^N \to \mathbb{R}^N$
∇	Discrete gradient operator	$\mathbb{R}^N \to \mathbb{R}^{d \times N}$
$\|\cdot\|_p$	$p-$norm of a vector	$\mathbb{R}^d \to \mathbb{R}$
$\mathcal{F}_{nD}[\cdot]$	$n-$dimensional Fourier Transform in Chapter 1	$L_2(\mathbb{R}^n) \to L_2(\mathbb{R}^n)$

0 Abstract

In diverse fields such as medical imaging, astrophysics, geophysics, or material study, a common challenge exists: reconstructing the internal volume of an object using only physical measurements taken from its exterior or surface. This scientific approach is called **tomography**, and is the foundational concept that motivates this work.

Specifically, the focus of my project is **divergent (or cone beam) X-ray tomography**, a technique used in medical imaging and material mechanics. Solving a tomography problems amounts to solving an **inverse problem**, where the ill-conditioning of the measurement operator, specifically the *X-ray transform operator*, and the inherent ill-posedness of inverse problems, *in the sense of Hadamard*, necessitate the integration of advanced optimisation techniques. Tackling these optimisation problems is highly challenging in practice due to the implementation of the *X-ray transform operator* and its mathematical adjoint, the *backprojection operator* : combination of **High Performance Computing** (HPC) and **exact adjoint match** between forward and backward operators has not been achieved in the currently available state-of-the-art packages.

During this project, the impact of an **adjoint mismatch** on the convergence of optimisation algorithms is addressed. Additionally, the implementation of forward and matched adjoint operators in both 2D and 3D is explored. Given that adjoint mismatches are common in many state-of-the-art tomographic libraries, we explore *whether it is worthwhile to approximate the adjoint instead of ensuring an exact match with the forward process for the sake of computational speed.*

Upon implementation, **reconstruction techniques** are investigated, from Bayesian formulations and prior hypotheses, to their corresponding representation problems in optimisation. Both **synthetic and real data**, in 2D and 3D, will be reconstructed using advanced techniques. Furthermore, we briefly explore novel applications unlocked with having consistent forward/adjoint codes, namely **uncertainty quantification** in scenarios with log-concave posterior distributions $p(x|y)$ of the reconstructed image x, given the measurements y. This exploration becomes particularly relevant when using a Maximum-A-Posteriori (MAP) estimate.

1 The X-ray transform : an introduction

1.1 Physics of X-ray absorption

An X-ray point source emits **high-energy photons** that penetrate a volumetric object. As these photons traverse the object, they interact with its internal structures, experiencing **energy attenuation** due to the object's absorption properties. The degree to which the photon energy decreases while traveling through the volume is dictated by the **Beer-Lambert law**, which describes the relationship between the absorption of light and the properties of the material through which the light is traveling. More formally, the Beer-Lambert law can be expressed as follows.

Beer-Lambert Law *(monochromatic)*

In a medium of homogeneous concentration c with a unique absorptivity coefficient ε,

$$\log\left(\frac{I_0}{I}\right) = \varepsilon l c$$

where I (resp. I_0) is the light intensity entering (respectively exiting) the medium with an optical path length of l.
More generally, for a heterogeneous medium with attenuation coefficient $\mu(\cdot)$,

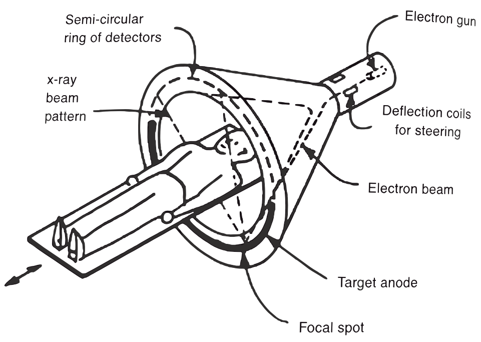
$$\log\left(\frac{I_0}{I}\right) = \int_0^l \mu(z)dz \tag{1.1}$$

where $\mu(\cdot) = \sum_i \epsilon_i c_i(\cdot)$.

In medical imaging, for example, a typical setup consists of a source that emits X-ray photons in a cone-beam pattern, coupled with detectors that capture the intensity of these photons (see [7] for more details). Practically, to obtain data y from a volumetric object represented by μ, we first measure the intensity of the photons without the object in place, denoted as I_0. Subsequently, we measure the intensity, I, when the object with an absorption coefficient μ is positioned in the path of the photon. If i denotes the detector index, the relation between y and μ can be written as:

$$\forall i \in [\![1...N]\!], \quad y^{(i)} = \log\left(\frac{I_0^{(i)}}{I^{(i)}}\right) = \int_{l^{(i)}} \mu(z)dz$$

where N represents the number of detectors and $l^{(i)}$ denotes the path of the ray from the source to the detector. The X-ray absorption model is useful for scanner imaging, PET scans, ultrasound imaging, and even cryo-EM.



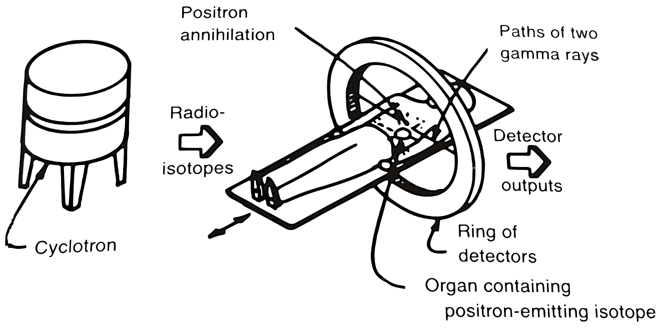


Figure 1.1: Practical physical setups for X-ray imaging (**left**) and PET imaging (**right**).

1.2 Mathematical definition and common properties of the X-ray transform

The operator associated with the general Beer-Lambert absorption law is the **X-ray transform**. Let Ω be a bounded domain in $\mathbb{R}^n$. Let f be a function from $\mathbb{R}^n$ to $\mathbb{R}$ such that f is compactly supported in Ω. The X-ray transform is the **integral of f across a straight line**.
Let us take $f \in L^1(\Omega)$ to ensure the existence of that integral. Let $\mathbb{S}^{n-1}$ be the unit sphere embedded in $\mathbb{R}^n$ of dimension $n-1$.

X-ray transform - generalised *(Definition)*

The X-ray transform X of f in the direction $\boldsymbol{\theta} \in \mathbb{S}^{n-1}$ is defined as follows:

$$\forall (\boldsymbol{x}, \boldsymbol{\theta}) \in (\mathbb{R}^n \times \mathbb{S}^{n-1}), \quad X_{\boldsymbol{\theta}}[f](\boldsymbol{x}) = \int_{t\in\mathbb{R}} f(\boldsymbol{x} + t\boldsymbol{\theta}) dt$$

$X_{\boldsymbol{\theta}}[f]$ is invariant to a translation of $\boldsymbol{x}$ in the direction $\boldsymbol{\theta}$. Therefore, a better parametrisation of this operator allows us to define a ray with a unique pair $(\boldsymbol{p}, \boldsymbol{\theta})$ in $(\mathbb{R}^{n-1} \times \mathbb{S}^{n-1})$.

$$\begin{aligned} X_{\boldsymbol{\theta}}[f] : \boldsymbol{\theta}^{\perp} &\to \mathbb{R} \\ \boldsymbol{p} &\longmapsto \int_{t\in\mathbb{R}} f(\boldsymbol{p} + t\boldsymbol{\theta}) dt \end{aligned}$$

where a ray is described by a directional vector for the line $\boldsymbol{\theta}$, and an offset $\boldsymbol{p}$ in the hyperplane span$(\boldsymbol{\theta})^{\perp}$.

Remark : The X-ray transform of a function *is not* the the Radon transform. The Radon transform corresponds to a volumetric integration across affine hyperplanes. The affine hyperplane is defined by a vector $\boldsymbol{\theta}$ setting up the hyperplane as span$(\boldsymbol{\theta})^{\perp}$, then a scalar p determines the affine translation of the hyperplane along the direction of $\boldsymbol{\theta}$. The X-ray transform is, in any dimension, an integration over **lines**. In that sense, the vector $\boldsymbol{\theta}$ defines the orientation of the line, and then a vector in the $(n-1)$-dimensional hyperplane span$(\boldsymbol{\theta})^{\perp}$ determines the affine translation.

In 2D, the two definitions coincide up to reparameterisation because the affine hyperplane is also a line.

2D case : X-ray / Radon transform *(Definition)*

In 2D, the X-ray transform X of f in the direction $\boldsymbol{\theta} \in \mathbb{S}^1$ is defined as follows :

$$\forall (p, \boldsymbol{\theta}) \in \mathbb{R} \times \mathbb{S}^1 \quad X_\theta[f](p) = \int_{t \in \mathbb{R}} f(p\hat{\boldsymbol{\theta}} + t\boldsymbol{\theta}) dt$$

where $\hat{\boldsymbol{\theta}}$ is the vector orthogonal to $\boldsymbol{\theta}$. Or equivalently,

$$\forall (p, \boldsymbol{\theta}) \in \mathbb{R} \times \mathbb{S}^1 \quad X_\theta[f](p) = \int_{\boldsymbol{x} \in \mathbb{R}^2} f(\boldsymbol{x}) \delta(p - \langle \boldsymbol{x}, \hat{\boldsymbol{\theta}} \rangle) \boldsymbol{dx}$$

where δ is the Dirac distribution.

Remark : $L_{p,\theta} = \{\boldsymbol{x} \in \mathbb{R}^2 \mid \boldsymbol{x} = \boldsymbol{p} + t\boldsymbol{\theta},\ t \in \mathbb{R}\}$ is the line along which the X-ray photons are traveling. We will refer to the function $p \mapsto X_\theta[f](p)$, for a given $\boldsymbol{\theta}$, as a **profile**. It represents the integration of the volumetric object across $\{L_{p,\theta}\}_{p \in \mathbb{R}}$, see Figure 1.2.

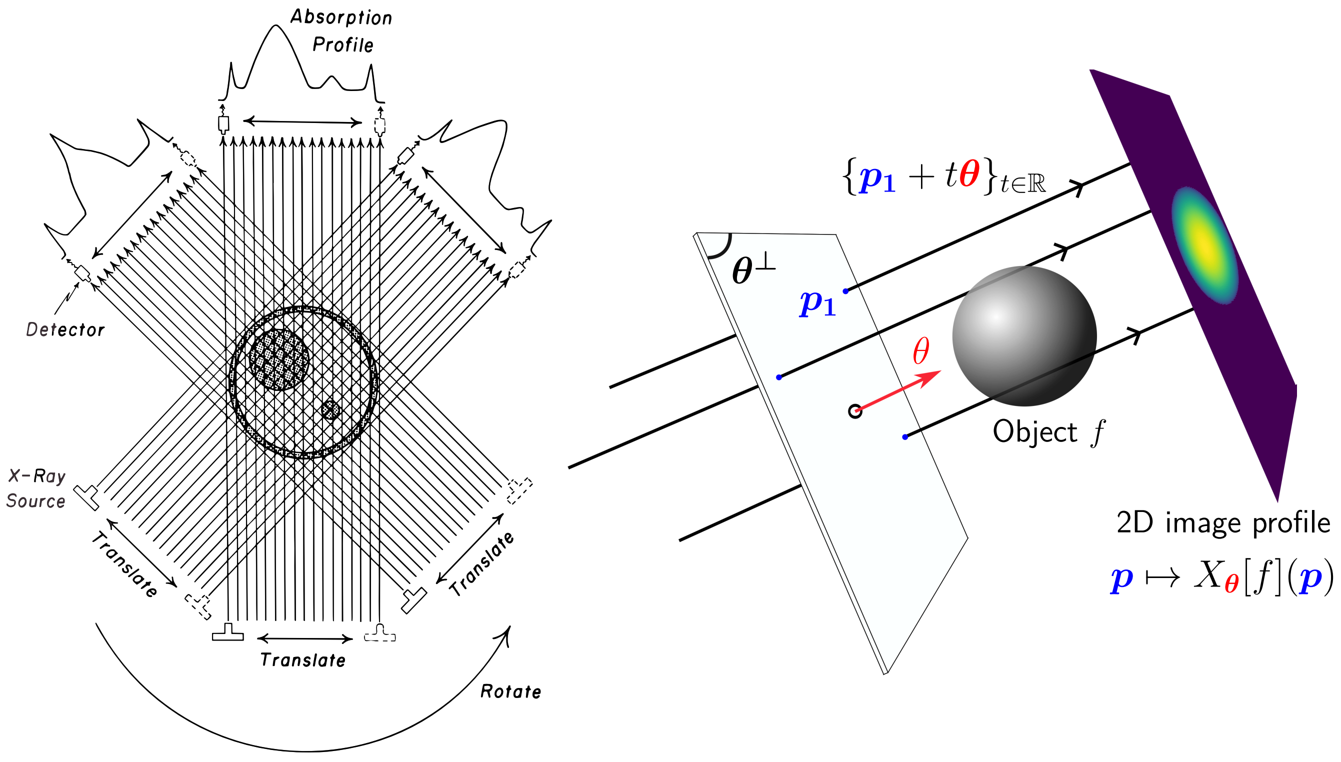

Absorption Profile
Detector
X-Ray Source
Translate
Translate
Translate
Rotate
$\{\boldsymbol{p_1} + t\boldsymbol{\theta}\}_{t \in \mathbb{R}}$
$\boldsymbol{\theta}^{\perp}$
$\boldsymbol{p_1}$
θ
Object f
2D image profile
$\boldsymbol{p} \mapsto X_{\boldsymbol{\theta}}[f](\boldsymbol{p})$

Figure 1.2: 2D visual representation of the X-ray transform of a basic shape. The absorption profiles are the functions $t \mapsto X_\theta[f](t)$ for three different angles θ. The variable θ represents the rotation parameter in experimental settings, while t represents the translation or offset parameter.

Basic properties can be derived from the definition of the X-ray transform. Iterative 1^st order methods use the forward and adjoint operators to update estimates, hence it is necessary to explicitly formulate the mathematical forward and adjoint definitions of the operator, as well as its inverse formula. One major property is the **convolution** property, which states that the X-ray transform of a convolution is the convolution of the X-ray transforms.

X-ray transform of a Convolution *(Property)*

For $f, g \in L^2(\mathbb{R}^n)$ and piecewise continuous,

$$X_\theta[f \star g] = X_\theta[f] \star X_\theta[g]$$

Proof: *Let $\boldsymbol{\theta} \in \mathbb{S}^{n-1}$ and $p \in \boldsymbol{\theta}^\perp$. Let $f, g \in L^1(\mathbb{R}^n)$. By definition,*

$$\begin{aligned}
X_\theta[f \star g](p) &= \int_{t\in\mathbb{R}} (f \star g)(\boldsymbol{p} + t\boldsymbol{\theta})\ dt = \int_{t\in\mathbb{R}} \left[\int_{x\in\mathbb{R}^n} f(x)g(\boldsymbol{p} + t\boldsymbol{\theta} - x)\ dx\right] dt \\
&= \int_{t\in\mathbb{R}} \left[\int_{(\tilde{p},\tilde{t})\in\boldsymbol{\theta}^\perp\times\mathbb{R}} f(\tilde{\boldsymbol{p}} + \tilde{t}\boldsymbol{\theta})g\left((\boldsymbol{p} - \tilde{\boldsymbol{p}}) + (t - \tilde{t})\boldsymbol{\theta}\right)\ d\tilde{p}\ d\tilde{t}\right] dt \quad \textit{(with X-ray parametrisation of } \boldsymbol{x}\textit{)} \\
&= \int_{(\tilde{p},\tilde{t})\in\boldsymbol{\theta}^\perp\times\mathbb{R}} \left[f(\tilde{\boldsymbol{p}} + \tilde{t}\boldsymbol{\theta}) \int_{t\in\mathbb{R}} g\left((\boldsymbol{p} - \tilde{\boldsymbol{p}}) + (t - \tilde{t})\boldsymbol{\theta}\right)dt\right]\ d\tilde{p}\ d\tilde{t} \quad \textit{(Fubini's theorem)} \\
&= \int_{\tilde{p}\in\boldsymbol{\theta}^\perp} \int_{\tilde{t}\in\mathbb{R}} f(\tilde{\boldsymbol{p}} + \tilde{t}\boldsymbol{\theta})\ X_\theta[g](\boldsymbol{p} - \tilde{\boldsymbol{p}})\ d\tilde{t}\ d\tilde{p} = \int_{\tilde{p}\in\boldsymbol{\theta}^\perp} \left[X_\theta[g](\boldsymbol{p} - \tilde{\boldsymbol{p}}) \int_{\tilde{t}\in\mathbb{R}} f(\tilde{\boldsymbol{p}} + \tilde{t}\boldsymbol{\theta})d\tilde{t}\right] d\tilde{p} \\
&= \int_{\tilde{p}\in\boldsymbol{\theta}^\perp} X_\theta[g](\boldsymbol{p} - \tilde{\boldsymbol{p}})\ X_\theta[f](\tilde{\boldsymbol{p}})d\tilde{p} \\
&= (X_\theta[g] \star X_\theta[f])(p). \quad \square
\end{aligned}$$

Linearity and Shifting *(Property)*

- The X-ray operator X is a **linear operator**. Let $f, g \in L^1(\mathbb{R}^n)$

$$\forall(\alpha, \beta) \in \mathbb{R}^2, \quad X[\alpha f + \beta g] = \alpha X[f] + \beta X[g]$$

  This property simply comes from the linearity of the integral.

- The X-ray transform of a translated function $f(\cdot - \boldsymbol{\alpha})$ is a translated X-ray transform of that function f. Let $(\boldsymbol{\theta}, \boldsymbol{p}) \in \mathbb{S}^{n-1} \times \boldsymbol{\theta}^\perp$.

$$\forall\boldsymbol{\alpha} \in \mathbb{R}^n, X_{\boldsymbol{\theta}}[f(\cdot - \boldsymbol{\alpha})](\boldsymbol{p}) = X_{\boldsymbol{\theta}}[f](\boldsymbol{p} - \Pi_{\boldsymbol{\theta}^\perp}\boldsymbol{\alpha}) \qquad (1.2)$$

  where $\Pi_{\boldsymbol{\theta}^\perp}(\cdot)$ corresponds to the orthogonal projection on span$(\theta^\perp)$.

  This comes from the definition of the X-ray operator : the contribution of the vector $\boldsymbol{\alpha}$ along $\boldsymbol{\theta}$ is canceled out by the integration in that direction. A simple change of integration variable concludes the proof.

Remark : The combination of these properties can be used to compute the X-ray transform of a function expressed in some basis of the functional space (pixels, splines, etc.).

1.3 Adjoint definition and inversion formula

In X-ray imaging tomography, data is acquired using the X-ray transform operator X. To recover the volumetric image from data, the first approach is to apply the inverse continuous operator X^{-1}.

Given the ill-posedness of the problem and the ill-conditioning of the operator, iterative optimisation algorithms will be crucial for a regularised reconstruction (see next chapter).

Fourier slice *(Theorem)*

Let $f \in L^2(\mathbb{R}^n)$ and $\boldsymbol{\theta} \in \mathbb{S}^{n-1}$.

$$\forall \eta \in \boldsymbol{\theta}^\perp, \quad \mathcal{F}_{(n-1)}[X_{\boldsymbol{\theta}}[f]](\eta) = \sqrt{2\pi}\mathcal{F}_n[f](\eta) \tag{1.3}$$

where $\mathcal{F}_n(\cdot)$ denotes $n-$dimensional Fourier Transform.

Interpretation : *The image data $\{X_{\boldsymbol{\theta}}[f](\boldsymbol{p})\}_{p\in\boldsymbol{\theta}^\perp}$ acquired by the X-ray transform along direction $\boldsymbol{\theta}$ corresponds, when applying an $(n-1)$ dimensional Fourier transform, to the volumetric object f on the hyperplane or "slice" $\boldsymbol{\theta}^\perp$ in the $n-$dimensional Fourier space. This is illustrated in Figure 1.3.*

Proof*: Let $f \in L^2(\mathbb{R}^n)$, $\boldsymbol{\theta} \in \mathbb{S}^{n-1}$, and $\eta \in \boldsymbol{\theta}^\perp$. Let us simplify the Fourier notations with $\widehat{\cdot}$:*

$$\begin{aligned}
\widehat{X_{\boldsymbol{\theta}}[f]}(\eta) &= \frac{1}{\sqrt{2\pi}^{(n-1)}} \int_{p\in\boldsymbol{\theta}^\perp} X_{\boldsymbol{\theta}}[f](p) e^{-i\langle \eta, p\rangle} dp = \frac{1}{\sqrt{2\pi}^{(n-1)}} \int_{p\in\boldsymbol{\theta}^\perp} \int_{t\in\mathbb{R}} f(p+t\boldsymbol{\theta}) e^{-i\langle\eta,p\rangle}\, dp\, dt \\
&= \frac{1}{\sqrt{2\pi}^{(n-1)}} \int_{x\in\mathbb{R}^n} f(x) e^{-i\langle \eta, \Pi_{\boldsymbol{\theta}^\perp} x\rangle} dx \qquad \textit{and } \langle \eta, \Pi_{\boldsymbol{\theta}^\perp} x\rangle = \langle \eta, x\rangle \textit{ since } \eta \in \boldsymbol{\theta}^\perp. \\
&= \frac{1}{\sqrt{2\pi}^{(n-1)}} \sqrt{2\pi}^n \hat{f}(\eta) = \sqrt{2\pi}\hat{f}(\eta). \qquad \square
\end{aligned}$$

Note: The factor $\sqrt{2\pi}$ arises from the difference in dimensions between the first Fourier transform, which is performed on an $n-1$ dimensional hyperplane, and the second Fourier transform, which is performed on $\mathbb{R}^n$.

Fourier methods for X-ray tomographic reconstruction rely on the Fourier slice theorem (Eq. 1.3). We refer the reader to [7] for more details on this subject. Nevertheless, since we have a sample of acquisition (discrete angles and detectors), the data is associated with a **non-uniform sampling grid** in the Fourier domain. Fast Fourier Transforms (FFT) to invert the problem become problematic since interpolation is required in this case. The focus of this work will be on ray methods, with numerical **ray tracing** of projection and backprojection (Eq. 1.4) operators. For this last section, the Fourier operator, in any dimension, will be alleged with the notaion $\widehat{\cdot}$.

Adjoint of X-ray operator (*Backprojection*)

Let $f \in L^2(\mathbb{R}^n)$. The X-ray operator:

$$\begin{aligned} X \;:\; L^2(\mathbb{R}^n) &\longrightarrow L^2(\mathbb{S}^{n-1} \times \mathbb{R}^{n-1}) \\ f &\longmapsto X[f] \end{aligned}$$

has as adjoint operator X^* :

$$\begin{aligned} X^* \;:\; L^2(\mathbb{S}^{n-1} \times \mathbb{R}^{n-1}) &\longrightarrow L^2(\mathbb{R}^n) \\ g &\longmapsto X^*[g] \end{aligned}$$

such that :

$$\forall x \in \mathbb{R}^n, \quad X^*[g](x) = \int_{\boldsymbol{\theta} \in \mathbb{S}^{n-1}} g_{\boldsymbol{\theta}}(\Pi_{\boldsymbol{\theta}^\perp} \boldsymbol{x}) d\boldsymbol{\theta}. \tag{1.4}$$

Interpretation : *Performing a backprojection of the data set $\{g_{\boldsymbol{\theta}}(\boldsymbol{p})\}$ involves spreading its values back along the path it was originally projected from. This process effectively redistributes the information throughout the volume domain and then sums up contributions from all directions $\boldsymbol{\theta}$. See Figure 1.4 for an illustration.*

Proof*: Let $f \in L^2(\mathbb{R}^n)$, and $g \in L^2(\mathbb{S}^{n-1} \times \mathbb{R}^{n-1})$.*

$$\begin{aligned} \langle X[f], g \rangle_{L^2(\mathbb{S}^{n-1} \times \mathbb{R}^{n-1})} &= \int_{\boldsymbol{\theta} \in \mathbb{S}^{n-1}} \int_{p \in \boldsymbol{\theta}^\perp} X_{\boldsymbol{\theta}}[f](\boldsymbol{p}) \; g_{\boldsymbol{\theta}}(\boldsymbol{p}) \; d\boldsymbol{\theta} \; d\boldsymbol{p} \\ &= \int_{\boldsymbol{\theta} \in \mathbb{S}^{n-1}} \int_{p \in \boldsymbol{\theta}^\perp} \left[\int_{t \in \mathbb{R}} f(\boldsymbol{p} + t\boldsymbol{\theta}) \; dt \right] \; g_{\boldsymbol{\theta}}(\boldsymbol{p}) \; d\boldsymbol{\theta} \; d\boldsymbol{p} \\ &= \int_{\boldsymbol{\theta} \in \mathbb{S}^{n-1}} \int_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) \; g_{\boldsymbol{\theta}}(\Pi_{\boldsymbol{\theta}^\perp} \boldsymbol{x}) \; d\boldsymbol{x} \; d\boldsymbol{\theta} \qquad (\forall \boldsymbol{\theta}, \{\boldsymbol{p} + t\boldsymbol{\theta}\}_{\boldsymbol{p} \in \boldsymbol{\theta}^\perp, t \in \mathbb{R}} = \mathbb{R}^n) \\ &= \int_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) \; X^*[g] \; d\boldsymbol{x} = \langle f, X^*[g] \rangle_{\mathbb{R}^n} \\ &\text{where } \forall x \in \mathbb{R}^n, \quad X^*[g](x) = \int_{\boldsymbol{\theta} \in \mathbb{S}^{n-1}} g_{\boldsymbol{\theta}}(\Pi_{\boldsymbol{\theta}^\perp} \boldsymbol{x}) d\boldsymbol{\theta}. \qquad \square \end{aligned}$$

Inversion formula (Filtered backprojection Theorem)

Let $g \in L^2(\mathbb{S}^{n-1} \times \mathbb{R}^{n-1})$ such that $g = X[f]$. Then,

$$f = \frac{1}{2\pi |\mathbb{S}^{n-1}|} X^*[g_{\text{filtered}}] \tag{1.5}$$

where

$$\widehat{g_{\text{filtered}}}(\xi) = |\xi| \hat{g}(\xi)$$

Interpretation : *Reconstructing the volume f from the data set $\{g_{\boldsymbol{\theta}}(\boldsymbol{p})\}$ involves performing a $|\xi|$ **high pass filtering** of the data in the Fourier domain, denoted by $|\xi|$, and then **backprojecting** this filtered data, hence the name Filtered BackProjection (FBP).*

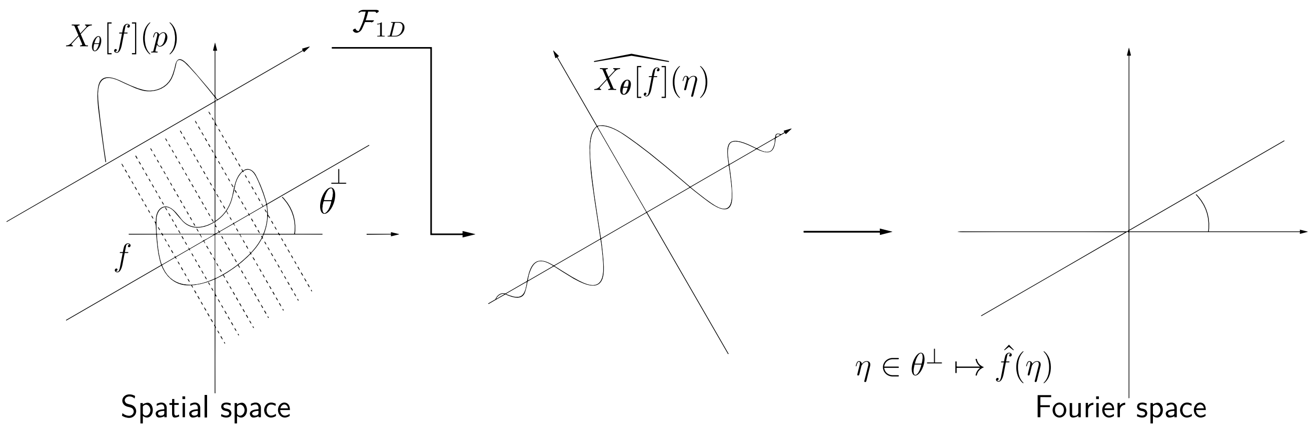


Figure 1.3: Illustration of the Fourier Slice Theorem 1.3

Remark : We can easily rearrange equation 1.5 and see that $X^*X[f] = h \star f$, with $\hat{h}(\xi) = \frac{1}{|\xi|}$ which is equivalent to saying that the operator X^*X returns a blurred version (with filter $\frac{1}{|\xi|}$) of the input image. This can be visualised, in figures 3.5 or 1.4.

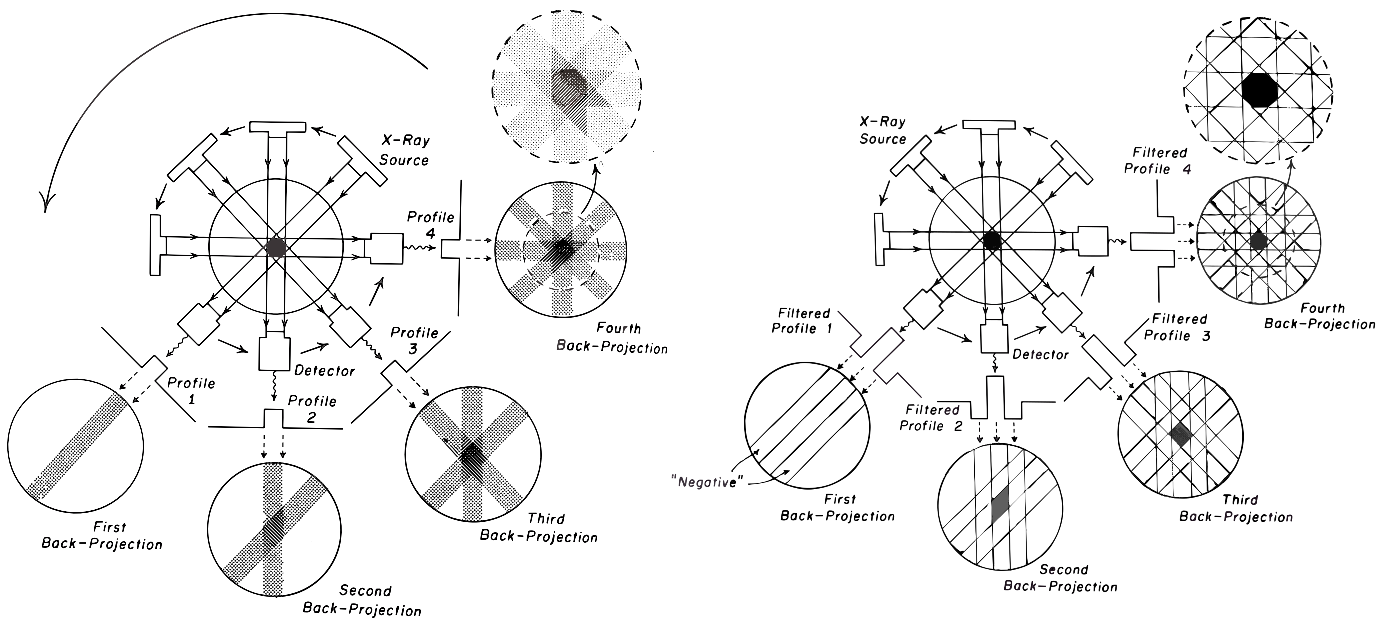


Figure 1.4: 2D visual representation of X-ray backprojection (left) and filtered backprojection (right). The acquired profiles $1...4$, which represent the function $t \mapsto g_{\theta}(t)$ for four different angles, are backprojected in the domain and summed up, without any filter (left) or after the $|\xi|$ high pass filter (right).

The circular arrow in Figure 1.4 represents the rotation of the X-ray source across different angles. For one given angle, the rays often travel either in a **parallel-beam geometry** (which is the case in Figure 1.4), or in a **cone-beam geometry** (ray traveling from a single point source to the detectors).

2 Inverse problems in tomography

2.1 Linear inverse problem formulation in the continuum

Now that the **acquisition operator** $X[\cdot]$ has been introduced in the previous chapter, we can now focus on correctly formulating the tomographic reconstruction problem. Keeping the previous notations, the objective is to reconstruct a volumetric object f – *for instance, the $3D$ density function of a brain.*

Inverse problem - Linear *(Definition)*

Let $\mathcal{H}$ be a Hilbert space, and let $f \in \mathcal{H}$.
Recovering the signal f from **noisy data measurements** $y \in \mathbb{R}^M$ acquired with the linear operator H is called a linear inverse problem :

$$\text{Find } f, \text{ s.t.} \quad y = Hf + n \tag{2.1}$$

$$(y_1, \ldots, y_M) = (\langle h_1, f\rangle_{\mathcal{H}}, \ldots, \langle h_M, f\rangle_{\mathcal{H}}) + n$$

where $H : \mathcal{H} \longrightarrow \mathbb{R}^M$ is the acquisition operator, n is random noise, typically Poisson or i.i.d Gaussian, and M is the number of acquisitions. The $(h_i)_{i\in[\![1\ldots M]\!]} \in \mathcal{H}^*$ (dual space of $\mathcal{H}$) are the sampling linear forms.

Using an X-ray scanner, a PET (Positron Emission Tomography) scan, or other setups, we only have access to acquisition data.

Remark (X-ray case) **:** To understand the definition in the context of X-ray tomography, let us explicitly define each mathematical object for this operator in 3D.

- In an X-ray setup, we measure M attenuations associated with each source/detector pair. This pair can be parameterised using the tuple $(\boldsymbol{p}, \boldsymbol{\theta})$ – where $\boldsymbol{\theta}$ represents the direction of the ray and $\boldsymbol{p}$ represents the offset in $\boldsymbol{\theta}^\perp$.
- The signal f to be reconstructed is generally a 3D density volume. It is practically compactly supported, so setting $\mathcal{H} = L^2(\mathbb{R}^3)$ is natural.
- Each of the M detectors measures a scalar, which is the integral of f along the ray path :

  $$\forall i \in 1 \ldots N, \quad y_i = X_{\boldsymbol{\theta_i}}[f](\boldsymbol{p_i}) + n_i$$

  with n being a random vector modelling the acquisition systematic noise typically.

- We can rewrite the sampling linear form $X_{\boldsymbol{\theta}_i}[f](\boldsymbol{p_i})$ as a scalar product :

$$X_{\boldsymbol{\theta}_i}[f](\boldsymbol{p_i}) = \int_{t\in\mathbb{R}} f(\boldsymbol{p}_i + t\boldsymbol{\theta}_i)dt = \int_{\boldsymbol{x}\in\mathbb{R}^3} f(\boldsymbol{x})\ \delta\left(\frac{\boldsymbol{x}-\boldsymbol{p_i}}{\|\boldsymbol{x}-\boldsymbol{p_i}\|} - \boldsymbol{\theta}_i\right) dx = \int_{\boldsymbol{x}\in\mathbb{R}^3} f(\boldsymbol{x})\ h_i(\boldsymbol{x})dx$$
$$= \langle f, h_i\rangle_{L^2(\mathbb{R}^3)}$$

where $h_i(\boldsymbol{x}) = \delta\left(\frac{\boldsymbol{x}-\boldsymbol{p_i}}{\|\boldsymbol{x}-\boldsymbol{p_i}\|} - \boldsymbol{\theta_i}\right)$ is the Dirac distribution that equals 1 along the geometric ray directed by the source-detector pair, and 0 otherwise.

2.2 Discrete formulation and ill-posedness

As explained in Section 2.1, in tomography, we have access to a **finite number of acquisitions**. Since we only capture M measurements, we can only reconstruct f with M degrees of freedom, motivating **parametric model** of f.

Discrete formulation of the inverse problem

Let $(\varphi_k)_{k\in 1\dots N} \in L^2(\mathbb{R}^n)$ such that :

$$\exists c_1 \dots c_N \in \mathbb{R}^N, \quad f = \sum_{k=1}^{N} c_k \varphi_k$$

Problem (2.1) now becomes :

$$\text{Find } \boldsymbol{c} = (c_k)_{k\in 1\dots N}, \text{ s.t.} \quad y = A\boldsymbol{c} + n \tag{2.2}$$

where $A \in \mathbb{R}^{M\times N}$ is the discrete operator matrix.
Indeed, keeping notations of (2.1)

$$\forall i \in 1\dots M \quad \langle h_i, f\rangle = \sum_{k=1}^{N} c_k \langle h_i, \varphi_k\rangle \tag{2.3}$$

so we can write the matrix A as follows :

$$A = \begin{bmatrix} \langle h_1, \varphi_1\rangle & \cdots & \langle h_1, \varphi_N\rangle \\ \vdots & \ddots & \vdots \\ \langle h_M, \varphi_1\rangle & \cdots & \langle h_M, \varphi_N\rangle \end{bmatrix} \tag{2.4}$$

In practice, this basis $(\varphi_k)_{k\in 1\dots N}$ is always chosen either (1) for its nice visual properties i.e. piecewise constant polynomial, piecewise continuous, or (2) for ites computational properties. In what follows we will mostly consider the X-ray transform for functions f where the (φ_k) are box functions (pixels in $2D$, voxels in $3D$) for computational reasons. See Appendix A for extensions of X-ray transform to Box-Splines parametric functions.

Remark (X-ray case) **:** In the 2D/3D case of the X-ray transform with pixel-discretisation, each coefficient of the matrix A represents the X-ray transform **for a specific ray** of the voxel. Since the voxel amplitude is constant and equals 1 in the corresponding hypercube, this matrix coefficient simply equals **the length of intersection between the ray and the voxel**.
Computational aspects will be discussed in detail in section 3, but we can already see here that to compute the X-ray transform of a pixelised volume f for a given ray (i fixed in Equation 2.3), we need the sum of all the voxel-ray intersections, weighted by the coefficients of f at the corresponding voxel. A major challenge is computing time. An efficient computation would only take into account the non-zero contribution voxels.

In theory, having $N = M$ (the same number of acquisitions as the number of coefficients to recover) would be ideal for solving the linear inverse problem, as it would allow the matrix A to be invertible. In practice, this is not the case, as there are many degenerate cases ($M \ll N$ for example).

Well-posed problem *(Hadamard definition)*

A mathematical problem is said to be well-posed, *in the sense of Hadamard* iff :

- The problem admits at least one solution *(Existence)*;
- The problem admits at most one solution *(Unicity)*;
- The solution depends continuously on the data *(Stability)*.

The last point has been particularly emphasised by Hadamard, and then by Hilbert (cf [3]) : *"the third requirement, particularly incisive, is necessary if the mathematical formulation is to describe observable natural phenomena. Data in nature cannot possibly be conceived as rigidly fixed; the mere process of measuring them involves small errors. Therefore a mathematical problem cannot be considered as realistically corresponding to physical phenomena unless a variation of the given data in a sufficiently small range leads to an arbitrary small change in the solution. This requirement of "stability" is not only essential for meaningful problems in mathematical physics, but also for approximation method"*.

The X-ray inverse problem is ill-posed in the sense of Hadamard.
Indeed, considering equation 2.2 without the random noise :

$$y = Hf \qquad (H \textit{ is the continuous X-ray operator})$$

then if the number of measurements is different from the number of unknowns, existence or uniqueness is not ensured. Nevertheless, we can hope for a pseudo-inverse solution, $\hat{f} = (H^*H)^{-1}H^*y$, which is the solution to the least-squares optimisation problem $\min_{f\in L^2} \|Hf - y\|_2^2$.
In this case, the stability condition is not verified :

$$\|\hat{f}\|_2 = \|(H^*H)^{-1}H^*y\|_2 \leq \frac{|\lambda_{max}(H)|}{|\lambda_{min}(H)|^2}\, y.$$

The term $\dfrac{|\lambda_{max}(H)|}{|\lambda_{min}(H)|^2}$ is unbounded when H is the X-ray transform operator since $\exists(\lambda_n)_n \in \mathbb{C}^{\mathbb{N}}$, a sequence of complex singular values such that (see[15]) :

$$\lim_{n\to\infty} |\lambda_n| = 0.$$

In this context, the X-ray operator is said to be **ill-conditioned** : the problem is ill-posed, even if the inverse of the X-ray transform can be derived analytically (see equation 1.5). To overcome this issue, the problem can be **regularised to ensure stability**.

2.3 From Bayesian formulation to regularised optimisation problem

Regularisation consists in injecting our *prior* knowledge of f in the objective function to penalise solutions which are not of the desired form. Solving $\hat{x} = \arg\min_{x\in\mathbb{R}^N} \mathcal{F}(Ax, y)$ is slightly modified with a regularisation term :

$$\widehat{x_\eta} = \arg\min_{x\in\mathbb{R}^N} \mathcal{F}(Ax, y) + \eta\mathcal{R}(x), \qquad \eta > 0. \tag{2.5}$$

For example, $\mathcal{R}(x) = \|x\|_2^2$, is one of the most common, corresponding to **Tikhonov regularisation**. When $\mathcal{F}(Ax, y) = \|Ax - y\|_2^2$, this regularisation can be seen as a small perturbation of the initial problem, and specifically affecting the **singular values** of $(A^*A)^{-1}$, **making them bounded**.

<u>Note</u>: The advantage of this regularisation term is the Morozov Principle result, stating that there exists a unique $\eta > 0$ such that

$$\|A\widehat{x_\eta} - y\|_2 = \delta \quad \text{where } \delta \text{ is the noise level of the data.}$$

The choice of $\mathcal{F}$ and $\mathcal{R}$ may seem arbitrary, but their form cam be formly chosen via **Bayesian formulation** of optimisation problems.

Bayes' Theorem

Let x and y be two probabilistic events with marginal probabilities $p(x)$ and $p(y) \neq 0$. Then :

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{2.6}$$

- The term $p(x|y)$ is called the posterior distribution.
- The term $p(y|x)$ is called the likelihood.
- The term $p(x)$ is called the prior distribution.
- The term $p(y)$ is called the marginal likelihood (or evidence).

In the context of tomographic inverse problems, y represents the observed data and x represents the corresponding reconstructed image :

$$y = Ax + n\ . \qquad \text{(from Eq. 2.2)} \tag{2.7}$$

When the posterior distribution $p(x|y)$ can be explicitly defined, we can solve the problem using an estimator for the unknown quantity x. One of the most common estimators is the **MAP** (*Maximum a Posteriori*), looking for the point x at which $p(x|y)$ is the highest. It amounts to solving the following optimisation problem :

$$\hat{x} = \underset{x \in \mathbb{R}^N}{\arg\min} -p(x|y)$$

or equivalently, since $\log(\cdot)$ is strictly increasing :

$$\begin{aligned} \hat{x} &= \underset{x \in \mathbb{R}^N}{\arg\min} - \log(p(x|xy)) && (2.8) \\ &= \underset{x \in \mathbb{R}^N}{\arg\min} - \log(p(y|x)) - \log(p(x)) && (2.9) \end{aligned}$$

since the evidence $p(y)$ is constant with respect to x.

- The likelihood distribution is the conditional probability of obtaining y given x. From equation 2.7, this corresponds to the **noise probability** since x is considered fixed in the likelihood. More precisely, since $n = y - Ax$,

  $$p(y|x) = p_{\text{noise}}(y - Ax)$$

  Modelling the noise by the common L-dimensional multivariate Gaussian process $\mathcal{N}(0, \Sigma)$:

  $$p(y|x) = \frac{1}{\sigma} \exp\left(-\frac{1}{2}(y - Ax)^T \Sigma^{-1} (y - Ax)\right) \tag{2.10}$$

  with σ the normalisation constant, and reinjecting $p(y|x)$ into equation 2.9, we get the log-likelihood term :

  $$-\log(p(x|y)) \propto \frac{1}{2} \|\Sigma^{-1/2}(y - Ax)\|_2^2 = \mathcal{F}(Ax, y)$$

  where we can clearly identify $\mathcal{F}$, the **data-fidelity** term of the cost function of equation 2.5. As we can see, **a least square term** comes from an i.i.d Gaussiane noise model.

- The prior distribution $p(x)$ is the information we infer based on our understanding of the volumetric object x. It often comes from either physics information or from a particular regular behavior towards which we want to guide the reconstruction. For example, if $p(x)$ is also Gaussian with a covariance matrix ηI_N, :

  $$-\log(p(x)) \propto \frac{\eta}{2} \|x\|_2^2 = \mathcal{R}(x)$$

  where we can clearly see the source of the **Tikhonov regulariser** as being due to **Gaussian priors**.

2.4 ASTRA toolbox: A leading state-of-the-art framework - features and limitations

Having defined the X-ray transform and formalised the reconstruction problem, our attention is naturally drawn to the identification of **reconstruction softwares** that performs **forward and backward X-ray operators** and, if possible, which include optimisation algorithms for efficient reconstruction. For the reconstruction of real experimental data, support for both 2D and 3D geometries is required, including commonplace cone-beam and parallel-beam setups. Finally, since Time-To-Solution (TTS) is crucial in computational imaging, high performance implementations targeting in CPUs and GPUs are important. Each community often has a domain-specific software, limiting spreading optimisation tools across imaging modalities. A non-exhaustive **benchmark** of the main Python-based packages for tomography is described in Table 2.1.

Table 2.1: Comparison of X-ray Transform software packages

Software	Ops [(1)]	HPC [(2)]	Beam geo. [(3)]	Method [(4)]	Regularisation [(5)]
ASTRA ↗	Fwd ↗, Adj ↗ (2D)	CPU + GPU ↗	Par/Fan (2D)↗ Par/Cone (3D)↗ (Slice) ↗	FBP, SART, SIRT, CGLS ↗	-
TomoPy ↗	Fwd ↗ -	GPU (SIRT, MLEM) ↗	Par (3D) (Slice) ↗	FBP, gridrec, SIRT ART, BART, MLEM ↗	TV, Tikhonov ↗
Scikit-image ↗	Fwd	-	Par (2D)	FBP, SART	-
SVMBIR ↗	Fwd ↗ Adj ↗	-	Par/Fan (3D) ↗	MBIR ↗ Plug&Play	σ_{MBIR} ↗
TOFU ↗	Fwd Adj ↗	CPU + GPU ↗	Par/Cone(3D) ↗ (Slice)	FBP (par.) FDK(cone) ↗	-
CT-Recon ↗	?	GPU ↗	Cone (3D) ↗	FDK ↗	-
RTK ↗	Fwd Adj ↗	CPU + GPU ↗	Cone (3D) ↗	CGD, SART FDK ↗	TV Wavelets ↗
PyHST2 ↗	Fwd ↗	CPU + GPU ↗	Cone ↗ Helical ↗	FBP (slice) ↗ FISTA ↗	TV, ↗ Wavelets, ...↗
Torch Radon ↗	Fwd Adj ↗	GPU ↗	Par/Fan (2D) ↗	Landweber ↗ CGD ↗	None
MATLAB ↗	Fwd ↗, Adj ↗	Yes ↗	Par/Fan (2D)	FBP ↗,↗	L2, TV
TIGRE	Fwd Adj	GPU	Flexible (2D, 3D) Par/Cone	Wide range	TV

[(1)] Operators : Fwd (forward X-ray transform), Adj (adjoint X-ray transform).
[(2)] HPC : High-performance computing resources, such as CPUs and GPUs.
[(3)] Beam Geometry : Parallel beam, cone beam, or both.
[(5)] Method : Type of algorithm used to reconstruct the image.
[(6)] Reg. : Regularisation techniques used to stabilise the reconstruction process.
↗ : External link to the reference page.

The **ASTRA Toolbox** package is undoubtedly the leading, widely used package for tomographic X-ray reconstruction. It incorporates X-ray forward and backward operators for cone-beam geometries and demonstrates impressive computational speed, particularly for 3D GPU-accelerated configurations. Nevertheless, **ASTRA has several drawbacks** :

- There are built-in optimisation algorithms, but they are not very flexible. No **custom regulariser or cost functionals** can be used, but only pre-defined optimisation algorithms can be executed.
- ASTRA ships with optimised parallel and cone-beam codes, but **random beam geometries** have very poor performance.(see section 3).
- The X-ray backprojection implementation, i.e. the adjoint, has been chosen to be **mismatched** with the forward operator on GPU :

  $$\langle Ax, y\rangle \neq \langle x, A^* y\rangle$$

  where A is the implemented X-ray transform and A^* is its adjoint. This adjoint mismatch is a trade-off between mathematical accuracy and speed (see [1]).

The adjoint match is the most concerning aspect of implementing state-of-the-art operators, which is the motivation behind this work. Indeed, **convergence** of usual convex **optimisation algorithms** property is not guaranteed in this case (cf [4]). For example, the convergence of the steepest descent method is not guaranteed (Shi, Wei, Zhang (2011); Elfving, H (2018)).

Steepest descent convergence condition *(Theorem)*

Solving the usual least squares problem :

$$\hat{x} = \underset{x\in\mathbb{R}^{\mathbb{N}}}{\arg\min}\, \|Ax - y\|_2^2$$

with the steepest descent method :

$$x^{k+1} = x^k + \omega\tilde{A}^*(y - Ax^k)$$

converges to a pseudo-inverse solution ($\tilde{A}^*Ax = \tilde{A}^*y$) if and only if :

$$0 < \omega < \frac{2\mathcal{R}\text{e }\lambda_j}{|\lambda_j|^2} \qquad \text{and} \qquad \mathcal{R}\text{e }\lambda_j > 0 \tag{2.11}$$

where the $\{\lambda_j\}_j$ are the singular values of $\tilde{A}^*A$.

In practice, when $\tilde{A}^*$ is unmatched with A, the second condition of 2.11 is not satisfied, which is unfortunately the case for most X-ray packages. When the projector/backprojector pairs are matched, the property 2.11 is verified, since A^*A **is symmetric positive-definite** :

$$\langle x, A^*Ax\rangle = \|Ax\|^2 > 0\ .$$

Furthermore, for advanced iterative methods, **adjoint mismatch is still a convergence issue**. [12] established the issue for Proximal Gradient Descent as well ; and Conjugate Gradient Descent, *one of the default ASTRA algorithms for reconstruction*, is obviously concerned by the mismatch : since $\tilde{A}^*A$ is not symmetric positive-definite, the underlying conjugation relation of the operator $\langle x, y\rangle_{\tilde{A}^*A} := \langle \tilde{A}^*Ax, y\rangle$ **does not define a proper inner product**.

To summarise, there are lots of **domain-specific** tools in the state-of-the-art of X-ray tomography packages. Some have high performance computing support on CPU and GPU, but there are still lots of custom **domain-specific algorithms**. For those reasons, it is hard to re-use those packages **across modalities**. Mathematical theory is also warning us of **convergence issues** if we use their **mismatched operators** in custom algorithms. Therefore, there is a need for **general-purpose** X-ray forward/backward code implementations which are **exactly matched**, to disseminate advanced methods faster and make them supplant current techniques.

3 Computational aspects

3.1 Introduction : Computational imaging goals

This section is centered around algorithms and software modules to implement the X-ray operator and its adjoint :

- for arbitrary 2D/3D geometries,
- on both CPU/GPU architectures,
- with constraint that it is the true adjoint to avoid convergence issues in iterative algorithms.

Enabling operability across **GPU and CPU** platforms enhances the versatility of our pipeline, ensuring the ability to tackle problems of all size.

In our journey to refine the computational process of tomographic reconstruction, the **implementation strategy** becomes paramount. Linear operator implementations can be broadly classified into 2 categories : matrix-based and matrix-free methods. **Matrix-based** methods compute and store the forward operator A, either dense or sparse format, then read it repeatedly to compute the operator. The adjoint in this case is simply given by the transpose, hence is readily available in matrix-based implementations. Their drawback however is **huge storage**, hence they do not scale well. **Matrix-free** methods on the other hand store an "algortihm" to evaluate the forward and the backward by leveraging the linear operator's structure. Prototypical Matrix-free operators are the Discrete Fourier Transform (DFT) and the "Sum" operator for which fast evaluation strategies exist : Fast Fourier Transform (FFT) and the reductions in 1D respectively. Matrix-free methods thus have the potential to be more efficient than their Matrix-free equivalents, and scale better with the size of the problem. Their drawback however is the **difficulty**, depending on the linear operator of interest, **in implementing the backward** (adjoint) for it to be consistent with the forward. Tomographic operators are clear ones where the Matrix-free forward implementation outperforms the Matrix-based counterparts by large margin on non-trivial problem sizes. We hence focus exclusively in what follows on Matrix-free approaches to evaluate the forward and backward operators.

Let us explain how the forward operator is computationally apprehended to understand both Matrix-based and Matrix-free approaches. We refer the reader to [10] for more advanced details. As explained in Equation 2.4, the operator A is expressed as :

$$A = \begin{bmatrix} \langle h_1, \varphi_1 \rangle & \cdots & \langle h_1, \varphi_N \rangle \\ \vdots & \ddots & \vdots \\ \langle h_M, \varphi_1 \rangle & \cdots & \langle h_M, \varphi_N \rangle \end{bmatrix} \in \mathbb{R}^{M \times N}$$

where N is the number of pixels of the image, M the number of rays, h_i is the X-ray linear form of the i^{th} ray and φ_j is the continuous representation of the j^{th} pixel. In that sense, each coefficient $\langle h_i, \varphi_j \rangle$ is the length of intersection between the ray i and the pixel j, as the pixel is constant equal to 1 on its support. By nature, the matrix A is very sparse, as each row (of size N, number of pixels) has only k non-zero coefficients, with k the number of pixels located along the ray path, so $k \leq 2\sqrt{N}$ (see Figure 3.1).

If $x \in \mathbb{R}^N$ is the image, the X-ray computation Ax with an explicit definition of A is extremely sub-optimal. Indeed, we prefer implementing a **matrix-free operator** to benefit the fact that the operator corresponds to an integration over a straight line (see Figure 3.1 for details). The image is naturally discretised in pixels here, but we can also innovate using splines to reconstruct piecewise polynomial - *and not just piecewise constant* - objects (see Appendix A).

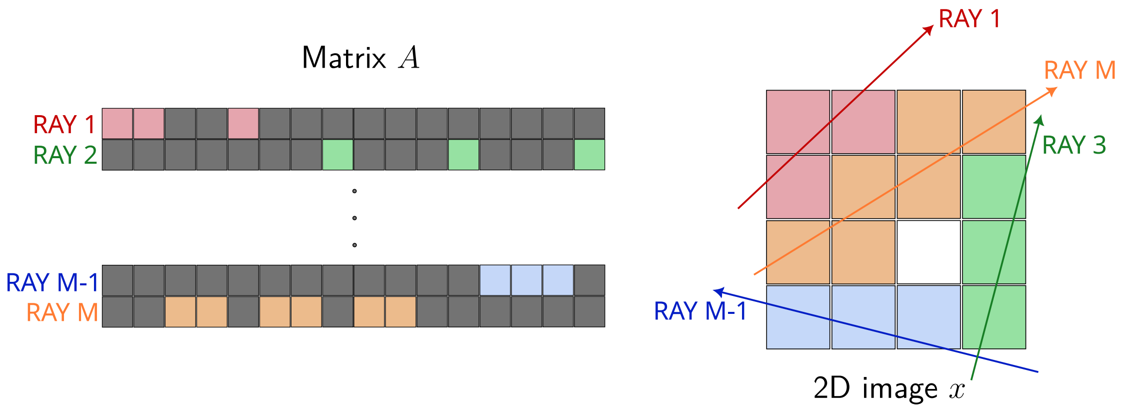


Figure 3.1: **Left :** Visualisation of the matrix $A \in \mathbb{R}^{M \times N}$ (N is the number of pixels of the image and M is the number of acquisitions). **Right :** Corresponding image $x \in \mathbb{R}^N$, with pixel colors corresponding to different ray pixel structures. To compute the `RAY 1` integral for example, instead of computing $(Ax)_1$, we prefer a matrix-free approach with an algorithm visiting each red pixel one by one in the direction of the ray, and calculating the weighted line-pixel intersections.

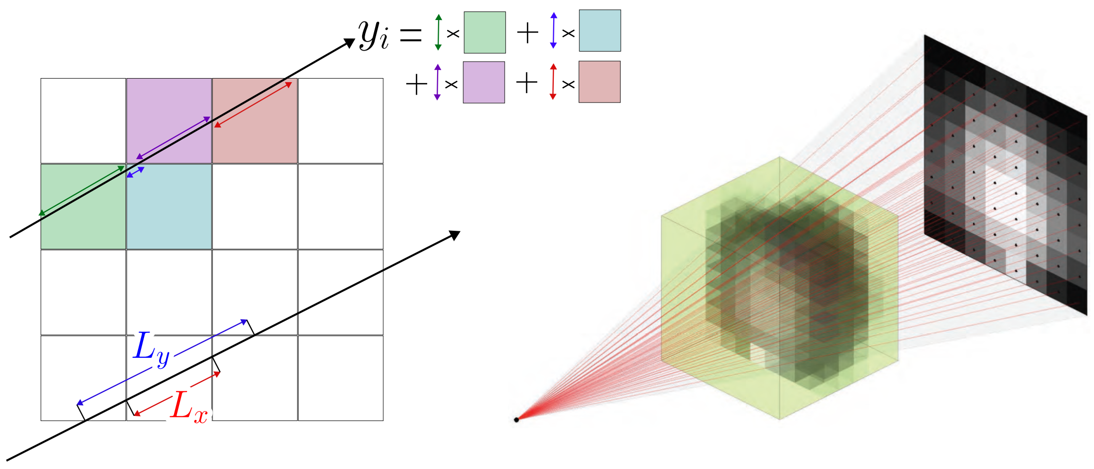
The most basic approach is to consider that the line-pixel intersection is approximately equal to 1 in any case. It amounts to considering that every non-zero coefficient of A is equal to 1. It is also a simpler approach because line-pixel intersection values are not required. However, since the goal is to compute an efficient and **accurate** X-ray transform, we need an algorithm to calculate each line-pixel intersection. State-of-the-art literature (cf. [6]) suggests the **Siddon's method** to be one of the most efficient. The key is finding a simple form to determine which pixels to cross given a line's orientation. Line-pixel intersection lengths are also estimated easily by tracking the residuals during the volumetric straight walk (line length remaining before crossing the next row), which is based on the distances L_x and L_y, equal to the distance the ray needs to cross two consecutive columns (or rows). Siddon's algorithm is a 2-step-process : (1) run Algorithm (1) to extract the output, `(i,j,length)` triplets which are the indices of crossed pixels and the corresponding line-intersection lengths ; and (2) read the required cells based on those triplets, and compute the weighted sum based on line lengths (cf Figure 3.2) :

```
projection = jnp.sum(lengths*image.at[indices]).
```

Algorithm 1 Siddon's Algorithm for line integrals in $2D$

```
Require: Image, Source position, Detector position (input).
Ensure: List of $K$ (i,j,length) triplets, $K$ being the number of crossed pixels (Output).
1:  Initialise $i, j = i_{\text{start}}, j_{\text{start}}$ and residual (:=length left before crossing next row)
2:  while $i \neq i_{\text{end}}$ or $j \neq j_{\text{end}}$ do
3:      length $= L_x$
4:      STORE($i, j$, length)
5:      residual $=$ residual $- L_x$
6:      $i = i + 1$
7:      if residual $< L_x$ then
8:          length $=$ residual
9:          STORE($i, j$, length)
10:         $j = j + 1$
11:         length $= L_x -$ residual
12:         STORE($i, j$, length)
13:         $i = i + 1$
14:         residual $= L_y - L_x +$ (residual)
15:     end if
16: end while
```

Remark (complexity) **:** The Siddon algorithm is expected to scale with the number of crossed pixels ($\sqrt{N}$ in 2D). This has been verified in the implemented algorithm.



$y_i = \updownarrow \times$ 🟩 $+ \updownarrow \times$ 🟦
$+ \updownarrow \times$ 🟪 $+ \updownarrow \times$ 🟥
L_y
L_x

Figure 3.2: **Left** : Priciple of Siddon's algorithm implementation. Ray on top represents how each data y_i data is computed with matrix-vector product $(Ax)_i$ corresponding to the sum of pixel-ray intersections weighted by the pixel values. The lower ray represents the main values used in the Siddon's algorithm L_x (resp. L_y), which is the length of the ray between two consecutive columns (resp. rows). **Right :** Illustration of the 3D discrete X-ray transform with cone-beam geometry.

Algorithm 1 describes Siddon's method for rays having a slope between 0 and 1. We note however that other cases can be handled with the same algorithm by transposing and flipping the image and ray coordinates. Initialising the parameters (`i_start`, `j_start`, `residual`) of algorithm 1 will not be discussed here. Our implementation of Algorithm 1 is Just-In-Time (JIT) compiled using JAX's [2] built-in routines, enabling efficient execution on CPU and GPU architectures. Algorithm 1 performs the X-ray transform for 1 source-detector pair, and we used it as a building block to perform it for cone-beam geometries (M rays, see Figure 3.3 for setup geometry visualisation at a given angle).

Finally, for CPU parallelism of the code, the code can be executed in **parallel using MPI**. The parallelism is applied to the outermost angles loop.

The code has been validated on test objects for which the cone-beam transform in known, i.e. spherical objects (Figure 3.4, with a disk in 2D), see Section 4.1 for details on the analytical projection of the sphere. We indeed observe a near-zero error, with residuals at the sphere boundaries due to pixelising the sphere to cope with box-basis assumptions.

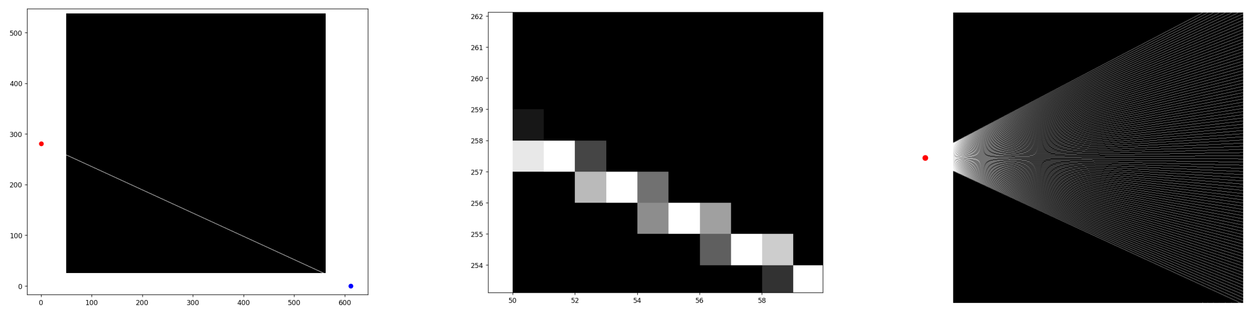


Figure 3.3: Visual results of our JAX-based Siddon line tracer. **Left**: Ray tracing from source (red) to detector (blue). **Middle**: Zoom on the line path. The gray-scale intensity $(0-1)$ of a pixel corresponds to the intersection length between the ray and the pixel. **Right**: Cone-beam discrete projection from a unique source (red) to equi-spaced detectors located on the right edge of the image.

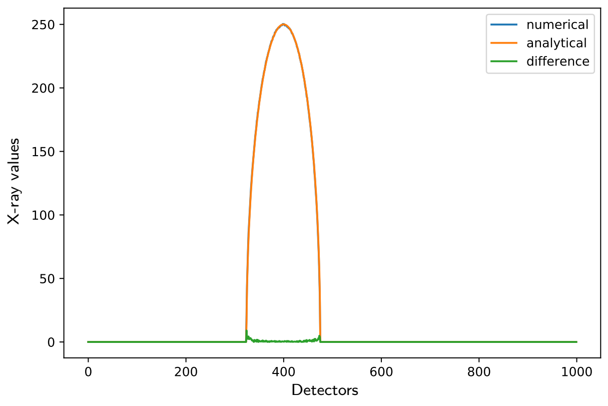


Figure 3.4: Comparison between analytical projection and path tracing numerical projection via Siddon's method. The orange curve corresponds to the analytical X-ray projection of a disk, and the blue curve corresponds to our Siddon algorithm applied to its pixelised equivalent.

3.2 Adjoint operator implementation

3.2.1 Auto-differentiation method

Now that the forward operator has been implemented, our focus is on the implementation of a **matched** adjoint, to be able to perform optimisation algorithms with **convergence guarantees**. At first sight, using JAX not only provides a support for code acceleration, but as a deep learning package it also provides auto-differentiation tools, such as automatic **vector-jacobian product**, which seems ideal in our case (see 3.1).

The adjoint computation is not a matrix-transpose calculation, so the code was initially implemented with the **JAX** package ([2]), a decision inspired by its inherent compatibility with **autodifferentiation**. The **vjp** (vector-Jacobian product) functionality embedded in JAX **paves the way for an exact calculation of the adjoint**. Indeed, when considering a Python function $f : \mathbb{R}^N \to \mathbb{R}^M$, the `vjp` tool of JAX enables us to automatically compute, for $(x, v) \in \mathbb{R}^N \times \mathbb{R}^M$:

$$(x, v) \mapsto \partial f(x)^T v$$

If f is the X-ray operator, linearity allows us to identify the operator $\partial f(x)$ with f itself. In that sense, the **vjp** of JAX can compute :

$$v \mapsto f^T v \tag{3.1}$$

Furthermore, JAX unveils an additional layer of efficiency through its **Just-In-Time compiling** feature (`JIT`). With the correct syntax in place, this feature enables code acceleration. JAX's support also extends to **automatic support for GPU**, marking a significant advancement in High Performance Computing.
To enable auto-differentiation tools, JAX requires a specific syntax to manage code loops. `While` and `For` loops are abandoned in favor of `scan` (function **`jax.lax.scan`**) loops. The main difficulty is that we need to **know in advance** the length of the loops when doing auto-differentiation, which is not the case in Siddon's method. For this reason, the loop sizes are overestimated by an upper bound, even if many iterations are obsolete.

The method has been **successfully implemented** despite significant struggles to express Siddon's method in JAX while retaining JIT and auto-differentiation functionality. Auto-differentiation rules ensure that the adjoint is matched with the forward. We verify this numerically by taking a random pair $(x, y) \in \mathbb{R}^N \times \mathbb{R}^M$ and compute the relative difference between $\langle Ax, y\rangle$ and $\langle x, A^*y\rangle$, obtaining relative errors of 10^{-5} to 10^{-7} depending on the image size. In comparison, **ASTRA** implementations have a relative difference mismatch of 10^1 to 10^3.

Nevertheless, this method appeared to be computing the adjoint operator orders of magnitude slower than ASTRA. To our understanding, this may be due to the fact that auto-differentiation needs to remember lots of state in the `Scan` loop to work, slowing down the adjoint computing time significantly. Even if this method appeared to be very interesting for specific problems and for educational purposes, we cannot rely on auto-differentiation to compute the adjoint efficiently for the X-ray transform, so we tried another method.

3.2.2 Custom matched algorithm

For the following, we ditch `scan` and go back to `while` loops since `while` is still JIT-compatible and easier to express X-ray transform with. To perform the **exact adjoint** of the X-ray transform, here is a reminder of what the adjoint is supposed to compute : The X-ray backprojection is about re-injecting the projection values into the image, along the ray direction. More precisely, for projection values $y \in \mathbb{R}^M$ acquired along specific rays, the backprojection is about computing, for each pixel φ_i :

$$\forall i \in 1...N, \qquad (A^T y)_i = \sum_{k=1}^{M} y_k \langle h_k, \varphi_i \rangle$$

where we recall that $\langle h_k, \varphi_i \rangle$ is the ray$^{(k)}$-pixel$^{(i)}$ intersection length. This expression means that ray k gives to pixel i the contribution of its intersection length with it, weighted by the data y_k. As for the forward operator, we prefer a matrix-free approach based on Siddon's algorithm. This time, **using again Siddon's algorithm** to get all indices of crossed pixels and their intersections (noted `(i,j,length)` in algorithm 1), the adjoint is computed using :

```
backprojection = jnp.zeros((n,n)).at[indices].add(lengths)
```

The results of X-ray projection and backprojection for cone-beam geometry are presented in Figure 3.5. As a reminder of Equation 1.5, FBP (Filtered BackProjection) is the theoretical inverse of the X-ray operator, corresponding to backprojecting a filtered version of the data.

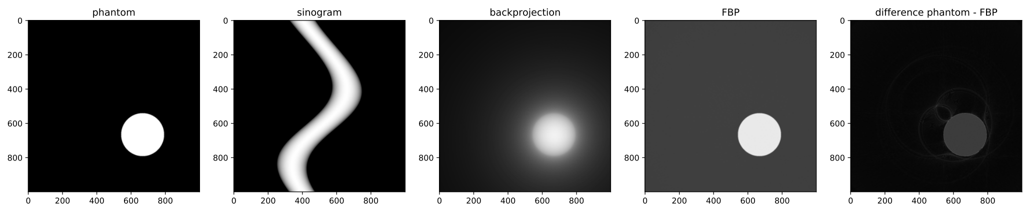


Figure 3.5: Visualisation of the implementation, from left to right : A disk phantom (image reference), the X-ray projection (called sinogram) of the disk, the adjoint X-ray backprojection of the sinogram, the naive reconstruction using Filtered BackProjection (FBP), and finally the difference between the phantom and the FBP reconstruction.

The table below (Figure 3.6) presents a benchmark of ASTRA 2D CPU code and our JAX 2D CPU code. Our code brings us to the conclusion that it is possible to compete with the state-of-the-art code, written in C++, when using JAX Just-In-Time compiling and MPI parallelism via custom adjoint rules.

	ASTRA	JAX	JAX + parallel	JAX AD version
Forward	0,9 min	2,5 min	0,8 min	62,8 min
Adjoint	0,9 min	2,5 min	1 min	74,5 min

Figure 3.6: Benchmark comparing our JAX code and ASTRA's CPU cone-beam code. Auto-differentiation is clearly too slow, and the parallelised version is close to ASTRA's performance. 2^d column is the custom adjoint version (`while` loops) and the last column is the `scan`-based auto-differentiation version.

3.3 3D generalised X-ray transform via rendering techniques

Siddon's method is unfortunately more difficult to generalise in 3D, and the method does not include the possibility of adding scattering effects efficiently if needed, which are common in real-world tomography. The idea behind the new 3D implementation comes from **computer graphics** :

- X-ray tomography needs **fine-grained parallelism** instead of coarse-grain (via MPI for instance), ao array APIs like Numpy or JAX are suboptimal.
- **Computer Graphics rendering** is an area where "ray"-based computations are omnipresent, hence their computing pipelines are optimized to exploit fine-grained parallelism efficiently. Fine-grained rays correspond to the fact that each ray can do very different computations from other rays (each voxel-ray intersection computation is independent).

To do so, we used the **Mitsuba** package. *Mitsuba 3 (see Article [8] and Package [9]) is a research-oriented retargetable rendering system, written in portable C++17 on top of the Dr.Jit Just-In-Time compiler. It is developed by the Realistic Graphics Lab at EPFL* .
With this method, **3D forward and adjoint** X-ray operators, for any geometry can be efficiently computed with high performance computing both in CPU and GPU. The adjoint is also **perfectly matched** with the forward. To perform projection or backprojection, the algorithm takes a list of the positions of the sources and the direction vectors of the rays. In the way the algorithm works, scattering effects can more easily be taken into account.

Pyxu (pronounced [piksu], formerly known as Pycsou) [14] is an open-source Python framework allowing scientists at any level to quickly prototype/deploy hardware accelerated and out-of-core computational imaging pipelines at scale.

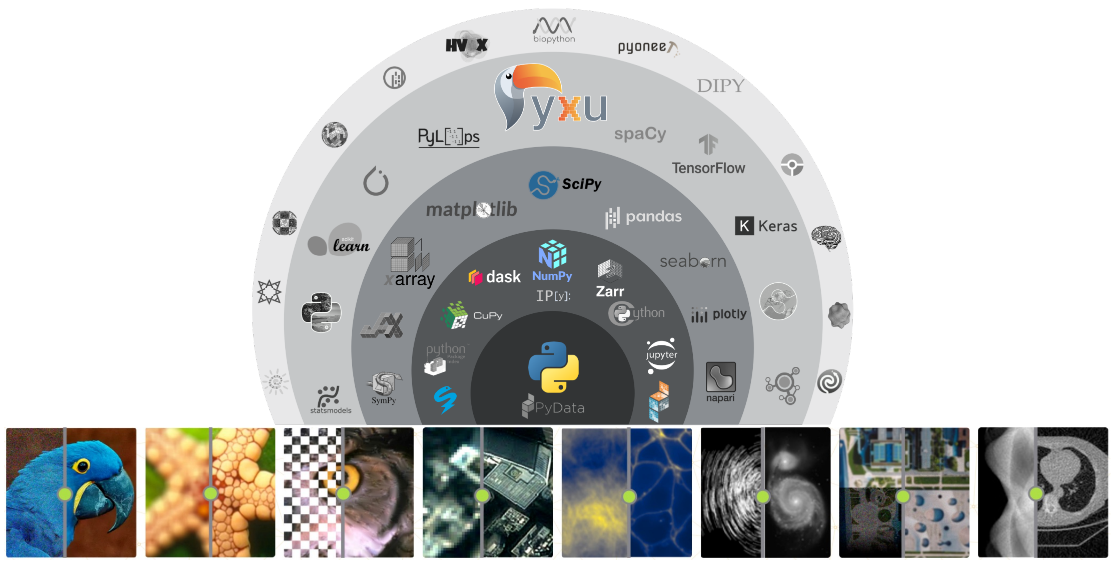


Figure 3.7: Pyxu framework illustration, managing many Python packages to perform computational imaging inverse problems, such as (left to right) image denoising, deblurring, inpainting, super-resolution, fusion, filtering and tomographic reconstruction.

Created by the Center for Imaging at EPFL, Pyxu enables the integration of **customizable regularisation methods** and **specific state-of-the-art algorithms** for solving computational imaging inverse problems. This grants researchers and users the freedom to effectively experiment with and optimise their reconstruction processes. This flexibility aims to overcome the limitations encountered with ASTRA. It allows tomographic reconstruction users to input their own regularisation algorithms, thus tailoring the reconstruction process to specific needs and datasets, enhancing both the quality and efficiency.

3.4 Performance evaluation & results

The validation of our JAx implementation of Siddon's method was limited to $2D$ due to complexity of implementing its $3D$ counterpart. The Mitsuba-based implementation is inherently $3D$ however, hence both $2D$ and $3D$ validations were performed. We note that the $2D$ X-ray transform is computed by embedding it in $3D$ setting without any performance degradation. We benchmarked against ASTRA cone-beam codes fo $2D$ and $3D$ setups on CPU and GPU in Table 3.1.

	Operation	Setup Dimension			
		2D, 250 CPU	2D, 2k CPU	2D, 250 GPU	2D, 2k GPU
ASTRA	FWD	0.14 s	71.6 s	1.91 ms	0.10 s
	BWD	0.11 s	94.1 s	2.47 ms	0.26 s
MITSUBA	FWD	0.22 s	12.5 s	6.26 ms	0.54 s
	BWD	0.11 s	10.8 s	5.73 ms	0.83 s
		3D, 300 CPU	3D, 150 CPU	3D, 150 GPU	3D, 300 GPU
ASTRA	FWD	$\times$	$\times$	20.4 ms	0.22 s
	BWD	$\times$	$\times$	18.8 ms	0.23 s
MITSUBA	FWD	29.4 s	1.33 s	82.7 ms	1.71 s
	BWD	33.3 s	2.23 s	120.0 ms	2.73 s

Table 3.1: Performance table of our Mistuba code benchmarked against ASTRA. FWD stands for forward operator and BWD stands for backward operator. Each column has a setup with the form `"nD, N, architecture"` : `nD` is the dimension of the image (2D or 3D), `N` is the number of pixels on each dimension (hence the total number of pixel is $\texttt{N}^n$) and the `architecture` is CPU or GPU. For each setup, we performed the X-ray transform with $\texttt{N}^n$ rays.

Note : GPU benchmarks are performed with 5 GPU warmup runs and computing times represent a mean of 10 code executions. We used the `benchmark` module from the `cupyx.profiler` package.The Graphic Card is the `NVIDIA RTX A5000` and the machine have 20 *CPUs. ASTRA does not have any CPU version of the 3D X-ray transform.*

The results show that the Computer-Graphics-based implementation is competitive with ASTRA cone-beam codes, with roughly a $4-8$ times speed difference in $3D$ GPU case. We however note that ASTRA code is heavily optimised for cone beam geometries, whereas our Computer Graphics implementation is not optimised for a specific layout. Optimising projection order to improve memory access patterns will certainly lead to sizeable performance gains and close the gap with ASTRA codes, while being adjoint matched. In CPU configuration, ASTRA does not take profit of multicore processing, so our code can be almost 9 times faster than ASTRA for a configuration of 20 CPUs available .

We now wonder what price ASTRA is paying in terms of mathematical accuracy for the convergence of its reconstruction methods. The JAX and Mitsuba codes have been embedded in a class inherited from the Pyxu Linear Operator class `LinOp`, **allowing reconstruction algorithm**. To highlight the adjoint mismatch in ASTRA, a Conjugate Gradient Descent reconstruction was carried out, using exactly the same algorithm twice, but once using the forward and backward operators of our JAX/Mitsuba code, and a second time using those of ASTRA. When using the CPU with ASTRA, the adjoint is matched, and the convergence and reconstruction results with our operator are very similar. On the other hand, **when using the GPU with ASTRA**, the adjoint is **no longer matched** (for computing time optimisation reasons), so there is no longer any reason for the algorithm to converge. The figure below illustrates this, highlighting the problem through the **relative error curve**, which is expected to decrease with the iterations, and through the **degradation of the image** with the iterations.

It is also important to note that 3D ASTRA operators are **only available with GPU support, so with a mismatched adjoint**.

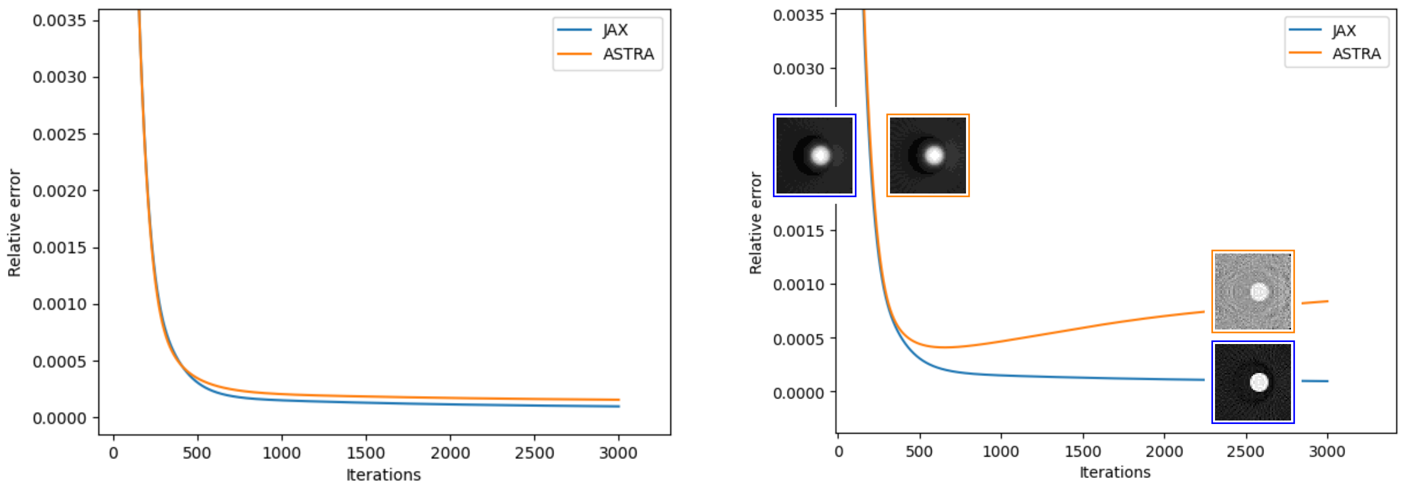


Figure 3.8: Conjugate Gradient reconstruction convergence curves. **Left:** ASTRA CPU vs JAX/Mitsuba relative error convergence curve. **Right:** ASTRA GPU vs JAX relative error convergence curve. ASTRA starts to diverge after several iterations due to the adjoint mismatch. Effects are also visible in the reconstruction images.

Hereafter, the phenomenon is again visualised with the succession of reconstructed images with different iteration steps. The Signal to Noise Ration (SNR) is exploding along iterations for ASTRA GPU reconstruction. The data has been generated with the analytical sphere projection detailed in section 4.1, and altered with i.i.d. random Gaussian noise.

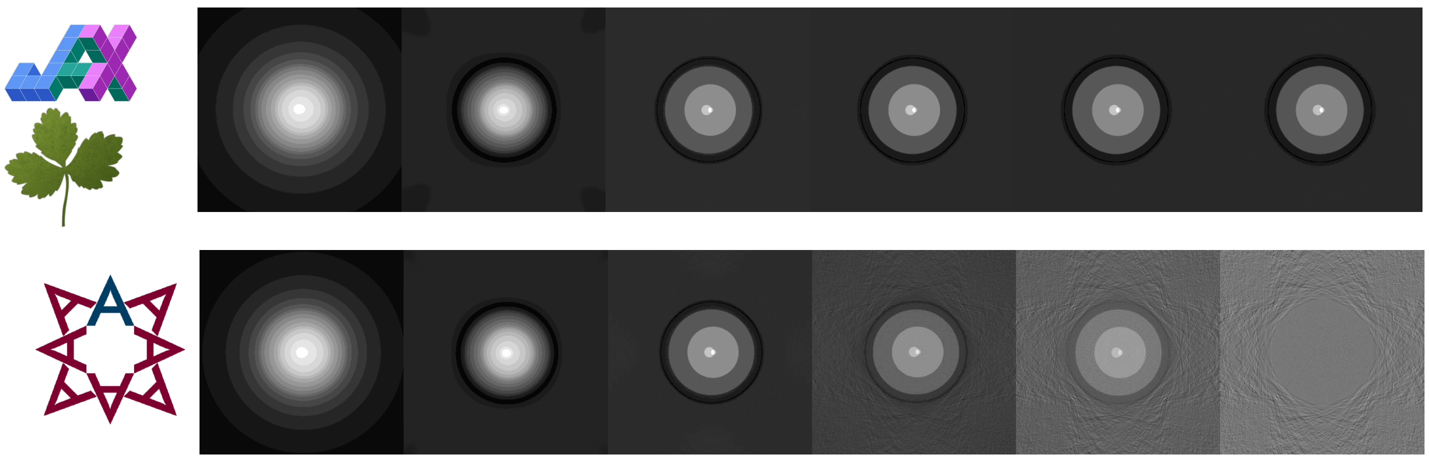


Figure 3.9: Reconstructed images with Conjugate Gradient across iterations (regularly spaced from 0 to 3000). The top row corresponds to our JAX code (same as Mitsuba code in 2D). The bottom row corresponds to the ASTRA 2D GPU reconstruction across iterations, showing that a systematic error is being accumulated over time. Not only the relative error, but also the cost function values, across iterations is divergent (see [13]).

In addition to its mathematical accuracy ensuring proper convergence, our code also allows users to achieve competitive runtimes (at worst few times slower than ASTRA GPU), but with **custom ray geometries**. Indeed, ASTRA optimises its code for **pre-defined geometries**, however when it comes to computing the X-ray operators for custom rays, ASTRA is really **under-optimised**: $68.5s$ for a small 3D setup $100 \times 100 \times 100$ and 100^3 rays, against $\approx 0.5s$ for our code. This necessity of not selecting a pre-defined geometry setup is the case in applications such as **microscopy** (where rays are not necessarily parallel or in a cone-beam geometry) or for **volumetric additive manufacturing** for example, when refraction of the material surface leads to **very specific geometries.**

It is useful for iterates to take place entirely on the GPU to avoid expensive CPU-GPU tranfers at each iteration. ASTRA **does not do this**, hence their performance advantage against Mitsuba is lost when the number of iterations is large. In that way, the optimisation algorithm's inputs and outputs would always stay stored in the GPU, so the transfer time between CPU and GPU would only take place once before the optimisation process, and once when the optimisation process is done. This is automatically the case for out implementation, whereas ASTRA is always returning CPU arrays, so in an iterative algorithm, there is at least K CPU-GPU transfers for K iterations. For large data, this transfer time accumulates over iteration steps. This memory management difference is illustrated if the right figure of Figure 3.10.

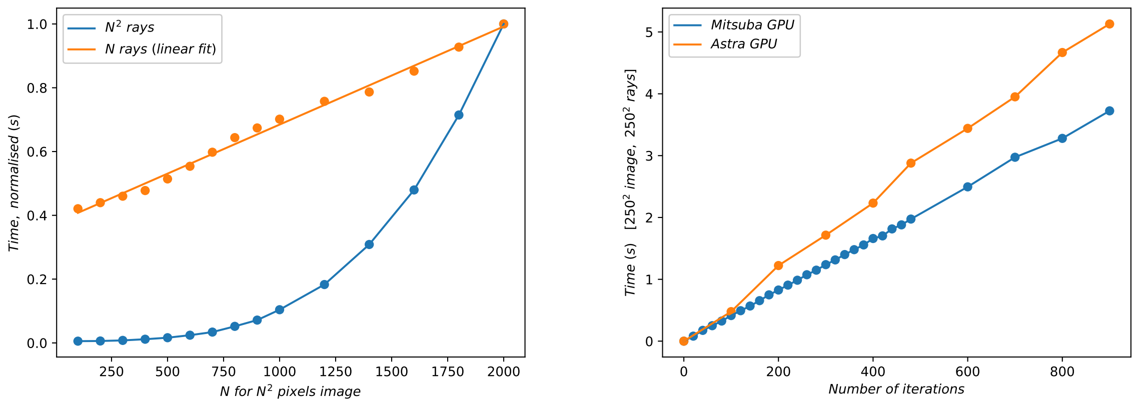


Figure 3.10: **Left:** Scalability checking of our code, linear with the number of rays. **Right :** Showing influence of CPU↔GPU transfer times for ASTRA and Mitsuba. Mistuba, unlike ASTRA, is transferring once the data at the beginning and at the end of the iterations.

4 Image reconstruction

4.1 Synthetic 3D data generation

X-ray operators in 2D and 3D have been implemented and their performance evaluated and compared to the ASTRA package. To use iterative algorithms and the EPFL-made package Pyxu (section 3), we need a reliable method to generate ground-truth data. Data can be generated using the forward implemented model itself, but it would be committing an inverse crime : reconstruction of data using the exact same model as the one used to generate the data. In order to generate synthetic X-ray projection data, generally in 3D, we can consider a homogeneous sphere, of center $\vec{c}_{sphere} = (a, b, c)$ and radius r :

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2.$$

Now, let us consider a ray of origin coordinates $(0, 0, 0)$ and direction $\vec{d}_{ray} = (u, v, w)$:

$$\mathcal{E} = \begin{cases} x = ut \\ y = vt \\ z = wt \end{cases} \qquad \textit{Parametric equation of the ray.}$$

Finding the two intersection points gives us the intersection length between the ray and the sphere. So the goal is to find t such that :

$$(ut-a)^2 + (vt-b)^2 + (wt-c)^2 = r^2$$

$$\Leftrightarrow t^2(u^2+v^2+w^2) - 2(au+bv+cw)t + (a^2+b^2+c^2-r^2) = 0.$$

This is equivalent to solving a 2^d order equation of parameters c_1, c_2, c_3 ; where $c_1 = ||\vec{d}_{ray}||^2$, $c_2 = -2\langle \vec{d}_{ray}, \vec{c}_{sphere} \rangle$ and $c_3 = ||\vec{c}_{sphere}||^2 - r^2$.

Figure 4.1 shows the image results of the code implementation of the above.

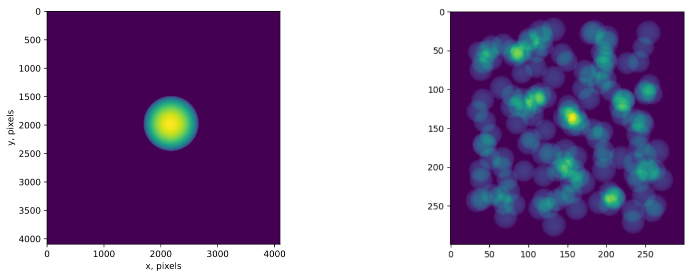


Figure 4.1: Visualisation of analytical sphere projections via cone-beam geometry, for one sphere (**Left**) and multiple randomly placed spheres (**Right**).

We used the **Numba JIT compiler** to improve data generation times. A **CUDA version** was also produced, although we noticed that the generation process is already very fast on the CPU. This code has contributed to a part of the *radioSphere* project[1].

4.2 Iterative algorithms for real data reconstruction

Now that synthetic data can be generated, we proceed to perform **full 3D reconstruction with regularisation.** After the data generation, we added some Gaussian noise to start apprehending real data problems. Optimisation theory can be quite complex in this case, but in this context, we can simplify to the essentials. The optimisation problem we are trying to solve is the following :

$$\hat{x} = \underset{x \in \mathbb{R}^N}{\arg\min}\, \mathcal{F}(Ax, y) + \eta \mathcal{R}(x), \qquad \eta > 0 \tag{4.1}$$

Here, since the noise is supposed to be Gaussian (cf Section 2.3),

$$\mathcal{F}(Ax, y) = \|y - Ax\|_2^2$$

Moreover, given our *a priori* knowledge that our image consist of spheres, sharp featured images (this statement is considered as a *prior*), the regularisation term we chose to incorporate is the **Total Variation** :

$$\mathcal{R}(x) = \mathsf{TV}(x) = \|\nabla x\|_1$$

which is supposed to **promote sparsity** of the gradient in the sense that it is penalising significant variations in the intensities of adjacent pixels and preserving edges while ensuring that only the uniform regions of an image are smoothed.

The cost functional has the form $\mathcal{F} + \mathcal{R}$ where $\mathcal{F}$ is differentiable and $\mathcal{R}$ is proximable. To solve this, we can use the Proximal Gradient Descent (PGD) using the package Pyxu. The `TV` class has been implemented, with an `apply` and a `prox` function, according to the Pyxu syntax for this algorithm.

Remark (differential Lipschitz constant)**:** A necessary parameter for the PGD algorithm is the **differential Lipschitz constant** of the cost function part $\mathcal{F}$. To do this, let us take $J_{\mathcal{F}}$ be the Jacobian of $\mathcal{F}$, and use the **power iteration technique** to compute its largest singular value, which is the Lipschitz constant when the operator is linear. Indeed, starting from a non-zero random vector b_0, we compute $b_{k+1} = \dfrac{J_{\mathcal{F}} b_k}{\|J_{\mathcal{F}} b_k\|}$, until the stopping criterion. Then, a good approximation of the Lipschitz constant is given by the convergence of the Rayleigh coefficient : $\dfrac{b_k^T J_{\mathcal{F}} b_k}{b_k^T b_k}$.

Reconstruction images in 3D with Proximal Gradient Descent algorithm are shown in the Figure 4.2.

[1]https://ttk.gricad-gitlab.univ-grenoble-alpes.fr/ttk/radioSphere

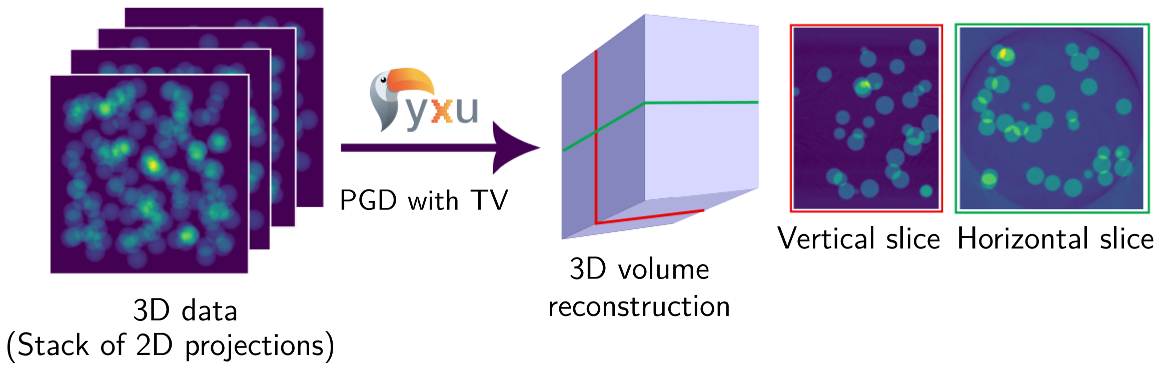


Figure 4.2: 3D spheres reconstructed using the PGD with Total Variation regularisation.

An experimental X-ray microtomography setup, named PIXE[2], is available at EPFL. At the core of the setup is a single stabilised axis of rotation, surrounded by an approximately 160° X-ray cone beam. The experiment table, on an air cushion, ensures extreme mechanical stabilisation, while a built-in correction mechanism adjusts for any potential misalignment of the rotation axis during acquisitions. It employs an electron gun, likely heated tungsten, that directs electrons with a $300\ nm$ accuracy. Two primary photoemission processes are used in this system. X-rays are converted to photons by a sensitive scintillator equipped with crystals to detect and direct the rays. Data is captured by a CCD (Charge-Coupled Device) located directly behind the scintillator. A picture of the setup is shown is Figure 4.3.

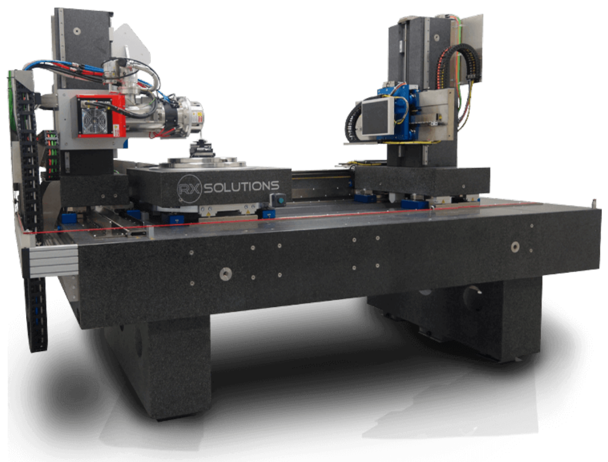


Figure 4.3: PIXE experimental setup, where the X-ray source, the flat screen detector and the rotating support are visible.

The data in this case if composed of multiple **solid sphere rocks** placed in a cylindrical tube. An experimental data acquisition provides us a stack of M_θ 2D images, where M_θ is the number of angles, and an `.xml` file describing the geometry setup (source-detector and source-

[2]https://www.epfl.ch/schools/enac/pixe/

object distances, pixel size, *etc*). In that way, **the code is adapted to the setup to allow reconstruction**. The 2D code and the 3D code have been tested with this data as follows :

- $2D$ reconstruction of 3D data. For cone-beam setup, we can consider only the $1D$ middle slice of each 2D projection data (see Figure 4.4). It that way, it is possible to reconstruct the $2D$ middle slice of the $3D$ volume. The reason is that the point source of X-rays is placed at the middle slice level of the object, so the rays passing through the object that are hitting the middle slice image detector are only crossing the 2D middle slice of the object.

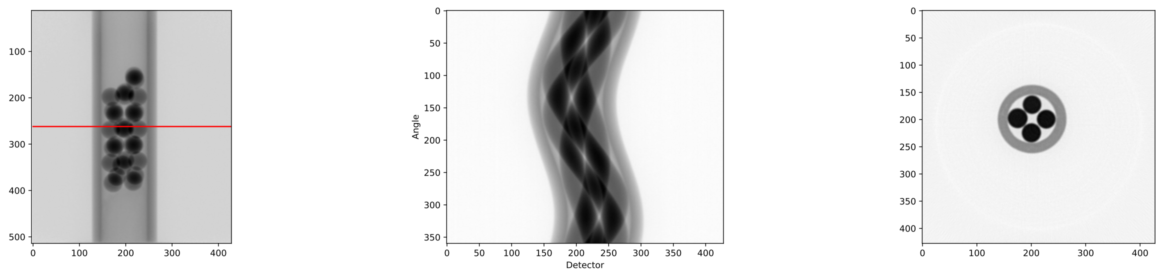


Figure 4.4: Reconstruction of the 2D data middle slice of the object. **Left :** 2D projection data at one given angle, and its $1D$ middle slice (red). **Middle** Stacked $1D$ slices for every angles (sinogram) corresponding to $2D$ data. **Right :** The reconstruction of the middle slice of the object, we can identify correctly the cylinder slice and the spheres inside.

- $3D$ full reconstruction of data. Once the previous preliminary step is done, full $3D$ reconstruction of real data have been done, with PGD algorithm and a TV regularisation (but small η regularisation parameter), visualised in Figure 4.5.

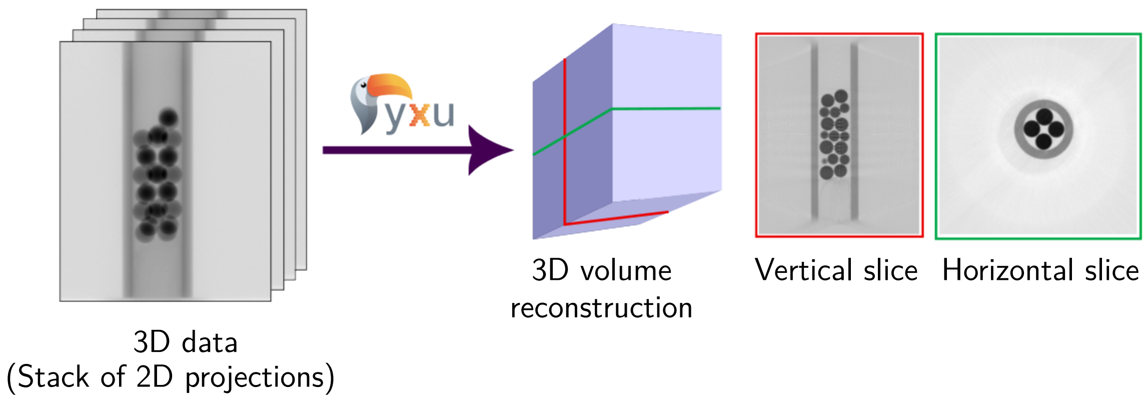


Figure 4.5: Reconstruction of the full $3D$ volumetric object. **Left :** Stack of projection images ($2D$) for every angle ($3D$ data). Reconstruction have been successfully performed and vertical and horizontal slices of the reconstruction are shown (**Right**).

4.3 Uncertainty quantification with confidence regions

Having the matched adjoint also allows us to perform algorithms requiring rigorous mathematical models. This is for example the case of **uncertainty quantification** which we chose to explore here. A simple approach of uncertainty quantification is our problem of recovering a signal $x \in \mathbb{R}^N$ from measurements $y \in \mathbb{R}^M$ with random noise vector $n \in \mathbb{R}^M$, with operator $A \in \mathbb{R}^{M \times N}$:

$$y = Ax + n.$$

The way we tried to solve the problem was to determine the **Maximum A Posteriori** (MAP) estimator of x:

$$\hat{x}_{MAP} = \arg\max_{x \in \mathbb{R}^N} p(x|y) = \arg\min_{x \in \mathbb{R}^N} \mathcal{F}(y, Ax) + \eta R(x)$$

with the same notations as in Equation 2.5, where the objective function is equivalent to the negative log-likelihood function of the noise distribution. The problem highlighted here is that **the MAP estimator gives a point estimate** $\hat{x}$, but it **does not provide any information about the uncertainty of the estimate.** The goal of uncertainty quantification is to provide a confidence region for the estimate $\hat{x}$. It could also help finding modelling parameters other than $\hat{x}$, for example the regularisation parameter.

Uncertainty quantification in this context involves **sampling the posterior** distribution of x given y. **Monte Carlo Markov Chain** methods, which are capable of constructing a Markov chain with a stationary distribution equal to the target distribution, are often employed. Despite their efficiency in handling low-dimensional distributions, their performance diminishes for high-dimensional ones.

An alternative approach, based on Marcelo Pereyra's **probability concentration inequality** theory (cf [11]), is applicable for log-concave distributions, including Gaussian, which is the case in our objective functional. It involves constructing a confidence region around the MAP estimator $\hat{x}_{MAP}$. This region, defined as

$$C_\alpha := \{\boldsymbol{x} : f(\boldsymbol{x}) + g(\boldsymbol{x}) \leq \gamma_\alpha\}$$

relies on the negative log-likelihood function $f = \mathcal{F}$, the penalising (or regularising) function $g = \eta\mathcal{R}$, and a threshold γ_α dependent on the confidence level α. With these notations, we have $p(x|y) = \exp(-f(x) - g(x))$.

Pereyra's Result on Probability Concentration Inequality [11]

The approximation of the confidence region C_α is provided by

$$C'_\alpha := \left\{\boldsymbol{x} \in \mathbb{R}^N : f(\boldsymbol{x}) + g(\boldsymbol{x}) \leq \gamma'_\alpha\right\}$$

with γ'_α an approximation of the Highest Probability Density threshold γ_α computed by

$$\gamma'_\alpha = f\left(\boldsymbol{x}_{MAP}\right) + g\left(\boldsymbol{x}_{MAP}\right) + \tau_\alpha \sqrt{N} + N,$$

and $\tau_\alpha = \sqrt{16 \log(3/\alpha)}$ as the universal constant.

One thing than can be done with this result is to construct an image where **each pixel represents a confidence range**, which is achieved by determining the largest value of each pixel of the MAP estimator such that the new image remains within the confidence region C'_α. Here is an example of this Uncertainty Quantification (UQ) experiment, reconstructing the MAP of a 50×50 image of a centered disk.

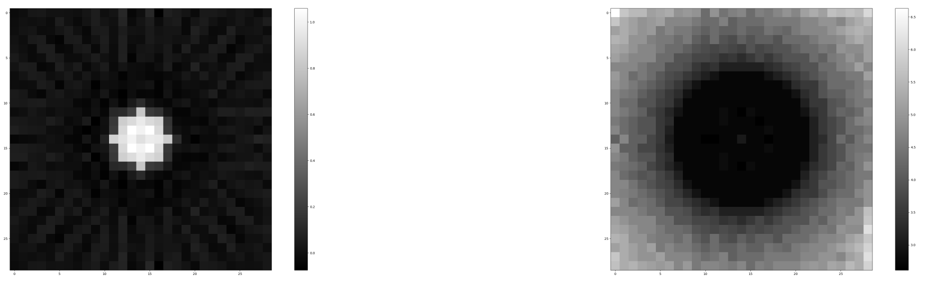


Figure 4.6: **Left :** MAP reconstruction estimation performed with Conjugate Gradient, with a least square data-fidelity term. **Right :** Pixel-wise uncertainty quantification using the method presented above.

Interpretation : For each pixel x_i of the MAP estimate x, the maximum value x_i^+ is computed, corresponding to the highest value such that x, where the i^{th} pixel is replaced by x_i^+ falls out of the confidence range C'_α. The same process is done with x_i^-, and the value of the UQ pixel-wise image is set to $x_i^+ - x_i^-$. Notably, the dark region on the right of the Figure 4.6 shows low uncertainty, correlated to the **geometrical disk where every cone beam ray intersects**, which is attributed to the specific acquisition geometry. Even so, **the results are very difficult to interpret.**

Conclusion

The mathematical analysis of the X-ray operator and its adjoint, but also of the iterative algorithms at stake in inverse problems in tomography, lead us considering to implement just-in-time compiled and GPU-supported codes to overcome the common adjoint mismatch issue appearing in the state-of-the-art packages. The results clearly show the possibility of the non-convergence of classical algorithms when using mismatched operators, whereas the convergence is guaranteed when using our codes. Moreover, the flexibility of the presented code, allows efficient computing on both CPU and GPU for any application domains, since computing time is weakly dependant to the ray geometry setup. Getting the adjoint matched also paves the way to more precise and accurate diagnostic tools in medical imaging, materials science, and other fields reliant on tomography. It is the case for uncertainty quantification, which could not be trusted if the MAP estimate is not even ensured to converge. The ongoing research could aim to explore the potential for integrating machine learning and deep neural networks algorithms to further enhance the accuracy, and reliability of tomographic reconstructions in specific domains of application.

A
Projection and backprojection with Box-splines : an overview

In order to represent and reconstruct **piecewise polynomial functions**, and not just piecewise constant ones, we need to discretise the volumetric object into different basis functions. A very practical basis function, namely **Box-Splines**, described in [5], allows to reconstruct those type of functions, getting us closer to the continuous representation of the object. The advantage of this method is that, for the Box-Splines functions β (represented in figure A.1), the **analytical X-ray transform** $X_\theta[\beta](p)$ is known. This is true for any degree of Box-Splines (order 0 classically reconstructs pixelised functions, 1^{st} order reconstructs picewise linear functions, *etc.*) The mathematical details are described in [5], and the implementation has been performed with a Numba-accelerated python code, storing an oversampled projection of the basis function in a *lookup table* (see Figure A.1 for the X-ray projections of a basis function). Then knowing the shifting property of the X-ray transform (see 1.2), we can use this *lookup table* to compute the projection and backprojection of a complete function expressed in the corresponding Box-Spline basis.

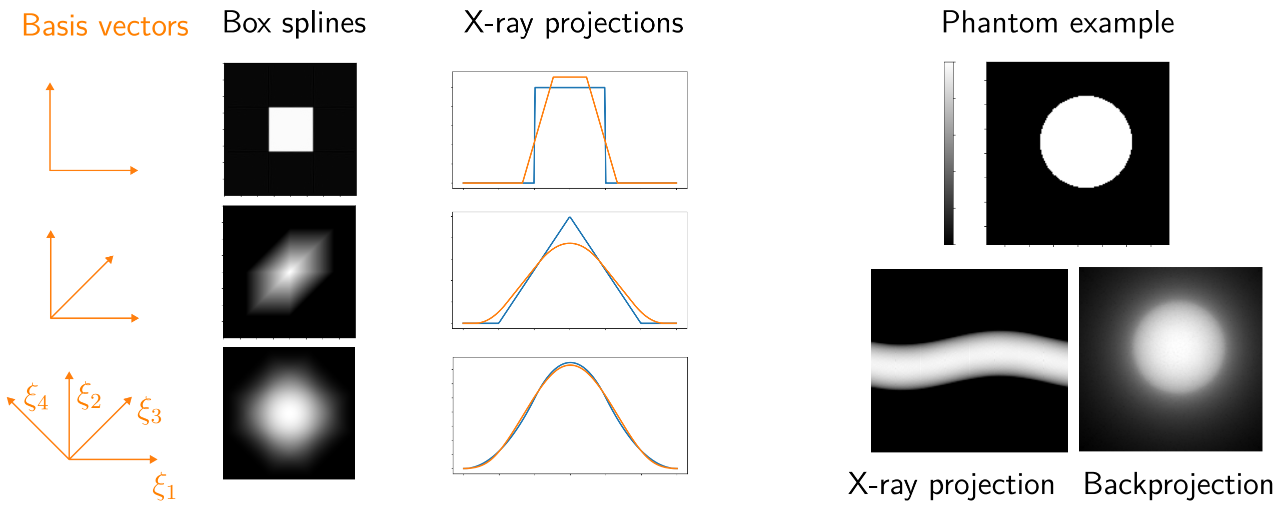


Figure A.1: Basis vectors (orange) generating Box-spline basis functions for image decompositions. The X-ray projections calculated analytically and sampled are represented (two different angles). On the right, a disk phantom, expressed as a decomposition into the 1^{st} degree Box-spline is being projected and then this projection is backprojected.

Bibliography

[1] *ASTRA Toolbox News*. 2022. URL: `https://www.astra-toolbox.com/news.html#news` (visited on 01/31/2022).

[2] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: `http://github.com/google/jax`.

[3] Richard Courant and David Hilbert. *Methods of Mathematical Physics*. Vol. 1. New York: Wiley, 1989. ISBN: 0585294283 9780585294285 9783527617210 3527617213.

[4] Tommy Elfving and Per Christian Hansen. "Unmatched Projector/Backprojector Pairs: Perturbation and Convergence Analysis". In: *SIAM Journal on Scientific Computing* 40.1 (2018), A573–A591. DOI: `10.1137/17M1133828`. eprint: `https://doi.org/10.1137/17M1133828`. URL: `https://doi.org/10.1137/17M1133828`.

[5] Alireza Entezari, Masih Nilchian, and Michael Unser. "A Box Spline Calculus for the Discretization of Computed Tomography Reconstruction Problems". In: *IEEE Transactions on Medical Imaging* 31.8 (2012), pp. 1532–1541. DOI: `10.1109/TMI.2012.2191417`.

[6] G. Han, Z. Liang, and J. You. "A fast ray-tracing technique for TCT and ECT studies". In: *1999 IEEE Nuclear Science Symposium. Conference Record. 1999 Nuclear Science Symposium and Medical Imaging Conference (Cat. No.99CH37019)*. Vol. 3. 1999, 1515–1518 vol.3. DOI: `10.1109/NSSMIC.1999.842846`.

[7] Per Christian Hansen, Jakob Sauer Jørgensen, and William R.B. Lionheart, eds. *Computed Tomography: Algorithms, Insight, and Just Enough Theory*. English. Society for Industrial and Applied Mathematics, 2021. ISBN: 978-1-611976-66-3.

[8] Wenzel Jakob et al. "Dr.Jit: A Just-In-Time Compiler for Differentiable Rendering". In: *Transactions on Graphics (Proceedings of SIGGRAPH)* 41.4 (July 2022). DOI: `10.1145/3528223.3530099`.

[9] Wenzel Jakob et al. *Mitsuba 3 renderer*. Version 3.1.1. https://mitsuba-renderer.org. 2022.

[10] Frank Natterer. *The Mathematics of Computerized Tomography*. USA: Society for Industrial and Applied Mathematics, 2001. ISBN: 0898714931.

[11] Marcelo Pereyra. "Maximum-a-Posteriori Estimation with Bayesian Confidence Regions". In: *SIAM Journal on Imaging Sciences* 10.1 (2017), pp. 285–302. DOI: `10.1137/16M1071249`. eprint: `https://doi.org/10.1137/16M1071249`. URL: `https://doi.org/10.1137/16M1071249`.

[12] Marion Savanier et al. "Proximal Gradient Algorithm in the Presence of Adjoint Mismatch". In: *2020 28th European Signal Processing Conference (EUSIPCO)*. 2021, pp. 2140–2144. DOI: `10.23919/Eusipco47968.2020.9287430`.

[13] Emil Y. Sidky et al. *Iterative image reconstruction for CT with unmatched projection matrices using the generalized minimal residual algorithm*. 2022. arXiv: `2201.07408 [physics.med-ph]`.

[14] Matthieu Simeoni et al. *matthieumeo/pyxu: pyxu*. DOI: 10.5281/zenodo.4486431. URL: https://doi.org/10.5281/zenodo.4486431.

[15] Wikipedia contributors. *Radon transform — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-October-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Radon_transform&oldid=1174924269.