

TIPE Transformée de Radon

Rouchouse Damien (24964)

2021 / 2022

- 1 Introduction
- 2 La reconstruction tomographique
- 3 La résolution discrète
- 4 Les résultats obtenus avec les codes Python
- 5 Annexe

Plan

1 Introduction

- La Transformée de Radon
- La loi de Beer-Lambert

2 La reconstruction tomographique

- Utilisation des sinogrammes
- La résolution analytique
 - La rétroprojection simple
 - La rétroprojection filtrée

3 La résolution discrète

- Une première méthode : la transformée de Fourier discrète
- Une deuxième méthode : reconstruction algébrique

4 Les résultats obtenus avec les codes Python

5 Annexe

- La transformée de Fourier
- La transformée de Fourier discrète
- Code Python : méthode par la transformée de Fourier discrète
- Code Python : méthode ART

Transformée de Radon

Transformée de Radon

Soit $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ une fonction continue à support compact. On définit la transformée de Radon par rapport à la droite paramétrée par (u, θ) par :

$$R[f](u, \theta) = \int_{-\infty}^{+\infty} f(u \cos \theta - v \sin \theta, u \sin \theta + v \cos \theta) dv$$

Plan

1 Introduction

- La Transformée de Radon
- La loi de Beer-Lambert

2 La reconstruction tomographique

- Utilisation des sinogrammes
- La résolution analytique
 - La rétroprojection simple
 - La rétroprojection filtrée

3 La résolution discrète

- Une première méthode : la transformée de Fourier discrète
- Une deuxième méthode : reconstruction algébrique

4 Les résultats obtenus avec les codes Python

5 Annexe

- La transformée de Fourier
- La transformée de Fourier discrète
- Code Python : méthode par la transformée de Fourier discrète
- Code Python : méthode ART

Lien avec la santé



Figure: Loi de Beer-Lambert

Lien avec la santé

Loi de Beer-Lambert

Soit (u, θ) le couple caractérisant la droite passant par le point $(u \cos(\theta), u \sin(\theta))$ et orthogonale au vecteur ${}^t(\cos(\theta), \sin(\theta))$

$$I(u, \theta) = I_0(u, \theta) \cdot \exp \left(- \int_{-\infty}^{+\infty} f(u \cos \theta - v \sin \theta, u \sin \theta + v \cos \theta) dv \right)$$

Lien avec la santé

Loi de Beer-Lambert

Soit (u, θ) le couple caractérisant la droite passant par le point $(u \cos(\theta), u \sin(\theta))$ et orthogonale au vecteur ${}^t(\cos(\theta), \sin(\theta))$

$$I(u, \theta) = I_0(u, \theta) \cdot \exp \left(- \int_{-\infty}^{+\infty} f(u \cos \theta - v \sin \theta, u \sin \theta + v \cos \theta) dv \right)$$

ou encore :

$$I(u, \theta) = I_0(u, \theta) \cdot \exp (-R[f](u, \theta))$$

Plan

1 Introduction

- La Transformée de Radon
- La loi de Beer-Lambert

2 La reconstruction tomographique

- Utilisation des sinogrammes
- La résolution analytique
 - La rétroprojection simple
 - La rétroprojection filtrée

3 La résolution discrète

- Une première méthode : la transformée de Fourier discrète
- Une deuxième méthode : reconstruction algébrique

4 Les résultats obtenus avec les codes Python

5 Annexe

- La transformée de Fourier
- La transformée de Fourier discrète
- Code Python : méthode par la transformée de Fourier discrète
- Code Python : méthode ART

Utilisation des sinogrammes

Définition

On appelle sinogramme la représentation des valeurs de la fonction $(u, \theta) \mapsto R[f](u, \theta)$ dans le plan d'axes u et θ .

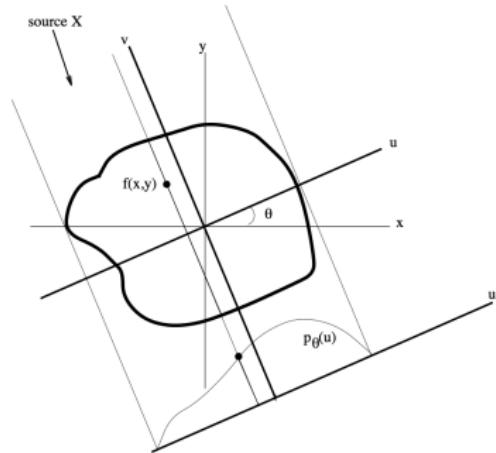
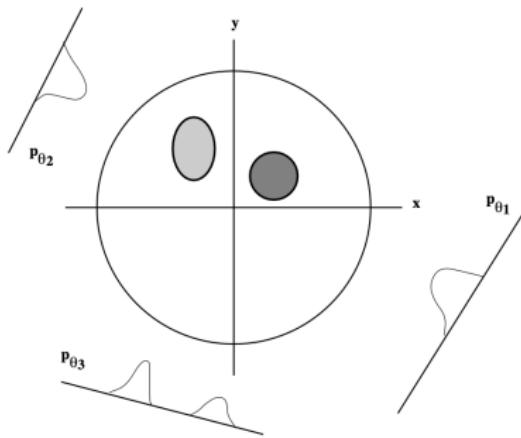
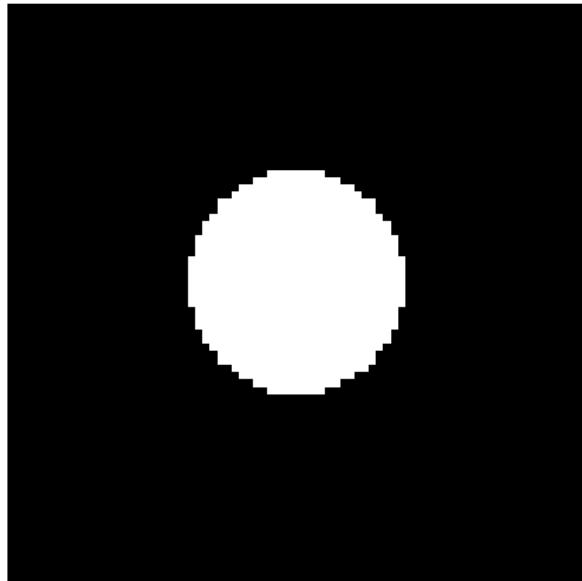


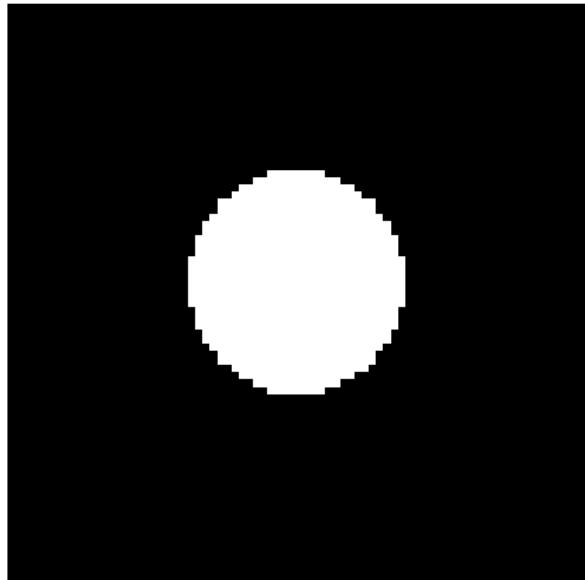
Figure: Projection

Utilisation des sinogrammes

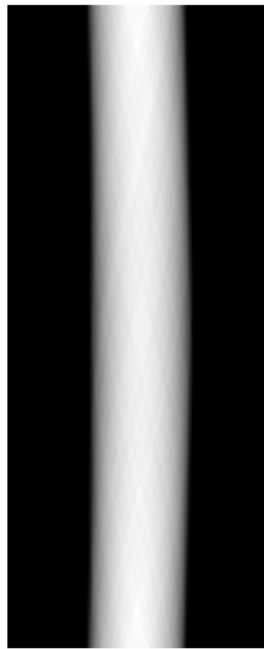


(a) Objet à reconstruire

Utilisation des sinogrammes



(a) Objet à reconstruire



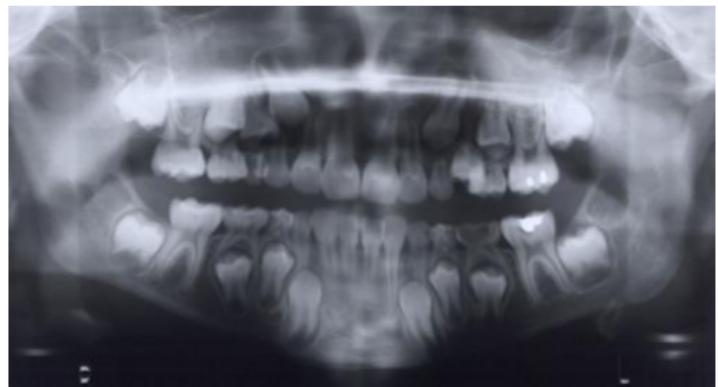
(b) Sinogramme de l'objet

Utilisation des sinogrammes

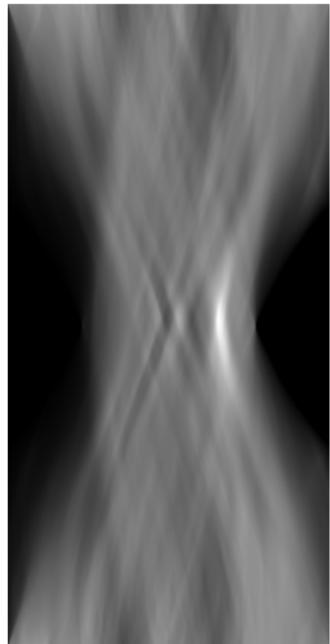


(a) Objet à reconstruire

Utilisation des sinogrammes



(a) Objet à reconstruire



(b) Sinogramme de l'objet

Utilisation des sinogrammes

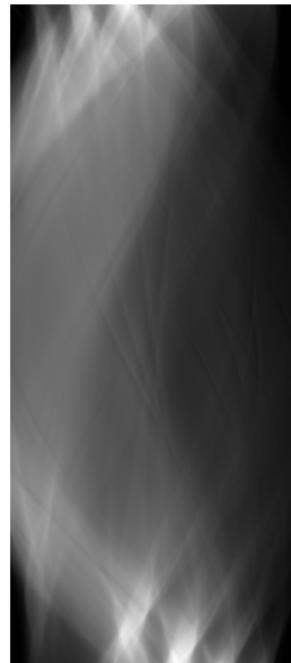


(a) Objet à reconstruire

Utilisation des sinogrammes



(a) Objet à reconstruire



(b) Sinogramme de l'objet

Plan

1 Introduction

- La Transformée de Radon
- La loi de Beer-Lambert

2 La reconstruction tomographique

- Utilisation des sinogrammes
- La résolution analytique
 - La rétroprojection simple
 - La rétroprojection filtrée

3 La résolution discrète

- Une première méthode : la transformée de Fourier discrète
- Une deuxième méthode : reconstruction algébrique

4 Les résultats obtenus avec les codes Python

5 Annexe

- La transformée de Fourier
- La transformée de Fourier discrète
- Code Python : méthode par la transformée de Fourier discrète
- Code Python : méthode ART

La reconstruction - Inversion analytique

Théorème de la coupe centrale

Soient $\theta \in [0, \pi[$ et $u \in \mathbb{R}$ et $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ la fonction caractéristique de notre objet. En notant $p_\theta : t \in \mathbb{R} \mapsto R[f](t, \theta)$ on a :

$$\hat{p}_\theta(u) = \widehat{f}(u \cos \theta, u \sin \theta)$$

où $\widehat{\cdot}$ est l'opérateur de transformée de Fourier.

Rétrécissement simple

Soit $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ la fonction caractéristique de notre objet. Pour $\theta \in \mathbb{R}$ on note $p_\theta : t \in \mathbb{R} \mapsto R[f](t, \theta)$. On a alors :

$$\forall (x, y) \in \mathbb{R}^2, \quad f(x, y) = \int_0^{2\pi} \int_{\mathbb{R}^+} \hat{p}_\theta(u) e^{2i\pi u(x \cos \theta + y \sin \theta)} du d\theta$$

La reconstruction - Inversion analytique

La rétroprojection simple (méthode naïve) :

objet à reconstruire

$$f(x, y)$$

La reconstruction - Inversion analytique

La rétroprojection simple (méthode naïve) :

objet à reconstruire

$$f(x, y) \xrightarrow{\text{projections}} \{p_\theta\}_\theta$$

La reconstruction - Inversion analytique

La rétroprojection simple (méthode naïve) :

objet à reconstruire

$$f(x, y) \xrightarrow{\text{projections}} \{p_\theta\}_\theta \xrightarrow{\text{Fourier}} \{\hat{p}_\theta\}_\theta$$

La reconstruction - Inversion analytique

La rétroprojection simple (méthode naïve) :



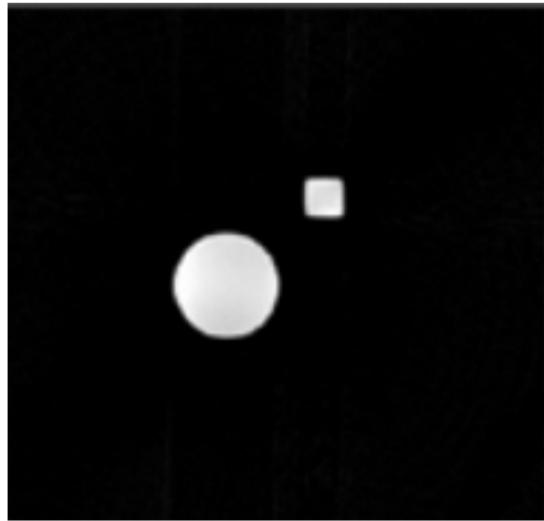
La reconstruction - Inversion analytique

La rétroprojection simple (méthode naïve) :



Inconvénient : problème de flou → nécessité de la rétroprojection filtrée

La reconstruction - Inversion analytique



La reconstruction - Inversion analytique

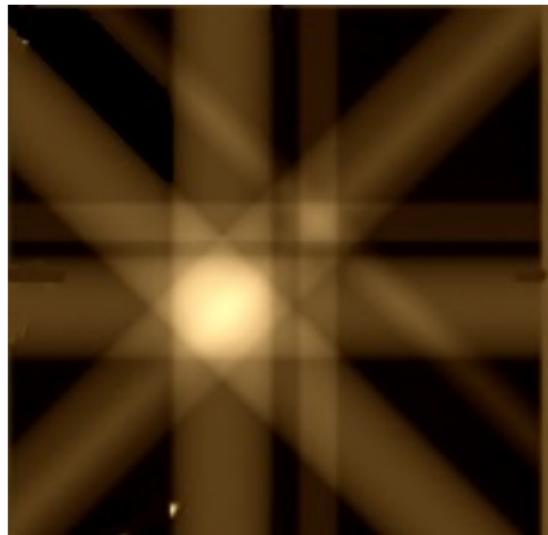
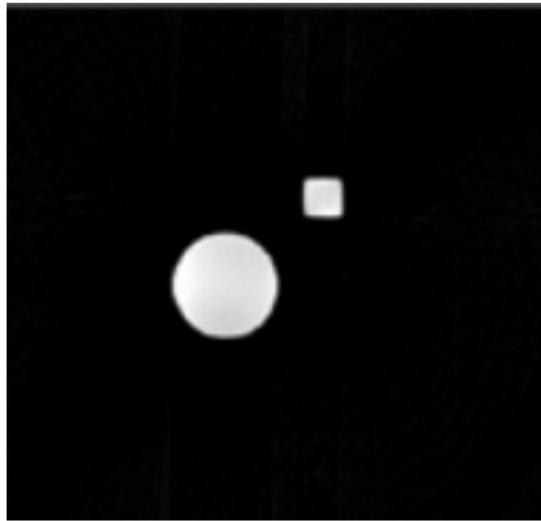


Figure: Rétroprojection simple

La reconstruction - Inversion analytique

→ Nécessité de l'utilisation des filtres

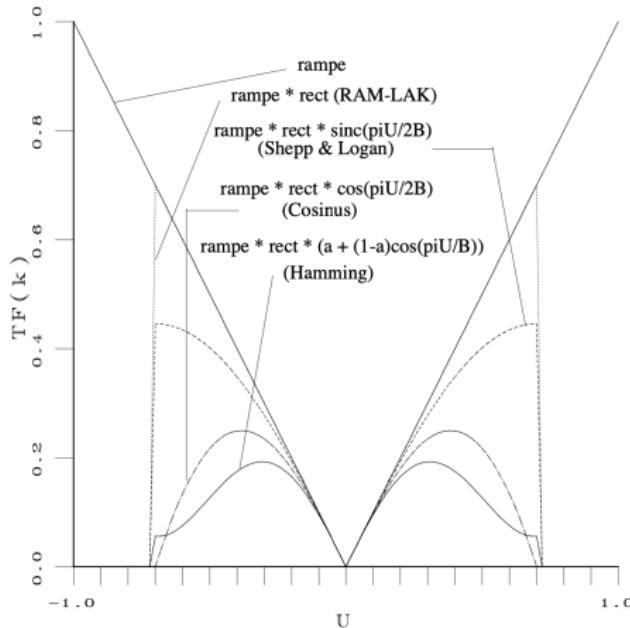


Figure: Quelques exemples de filtres

La reconstruction - Inversion analytique

Rétrorprojection filtrée

Soit $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ la fonction caractéristique de notre objet. Pour $\theta \in \mathbb{R}$ on note $p_\theta : t \in \mathbb{R} \mapsto R[f](t, \theta)$. On a alors :

$$\forall (x, y) \in \mathbb{R}^2, \quad f(x, y) = \int_0^{2\pi} \int_{\mathbb{R}^+} \hat{p}_\theta(u) |u| e^{2i\pi u(x \cos \theta + y \sin \theta)} du d\theta$$

La reconstruction - Inversion analytique

objet à reconstruire

$$f(x, y)$$

La reconstruction - Inversion analytique

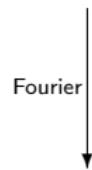
objet à reconstruire

$$f(x, y) \xrightarrow{\text{projections}} \{p_\theta\}_\theta$$

La reconstruction - Inversion analytique

objet à reconstruire

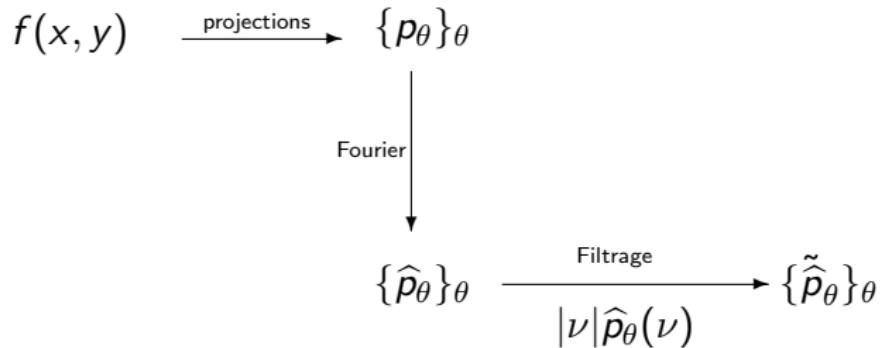
$$f(x, y) \xrightarrow{\text{projections}} \{p_\theta\}_\theta$$



$$\{\hat{p}_\theta\}_\theta$$

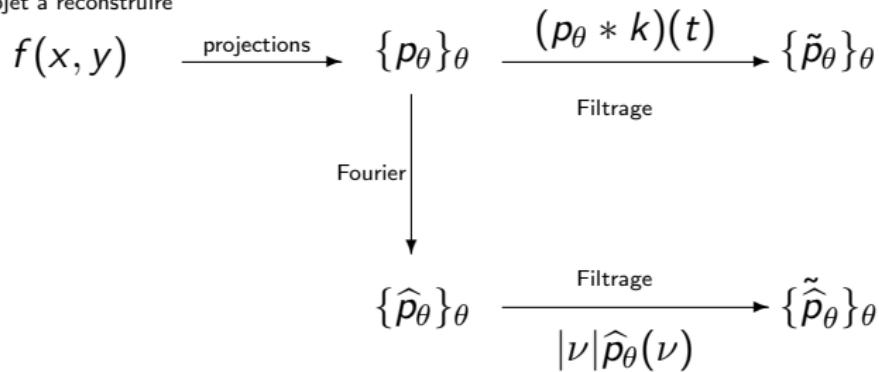
La reconstruction - Inversion analytique

objet à reconstruire



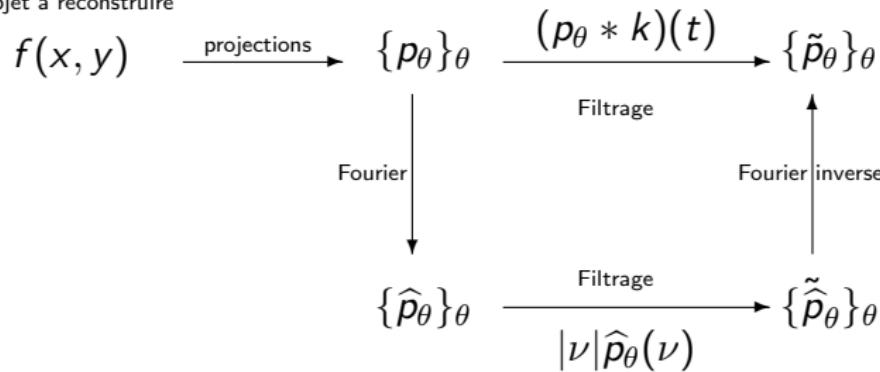
La reconstruction - Inversion analytique

objet à reconstruire

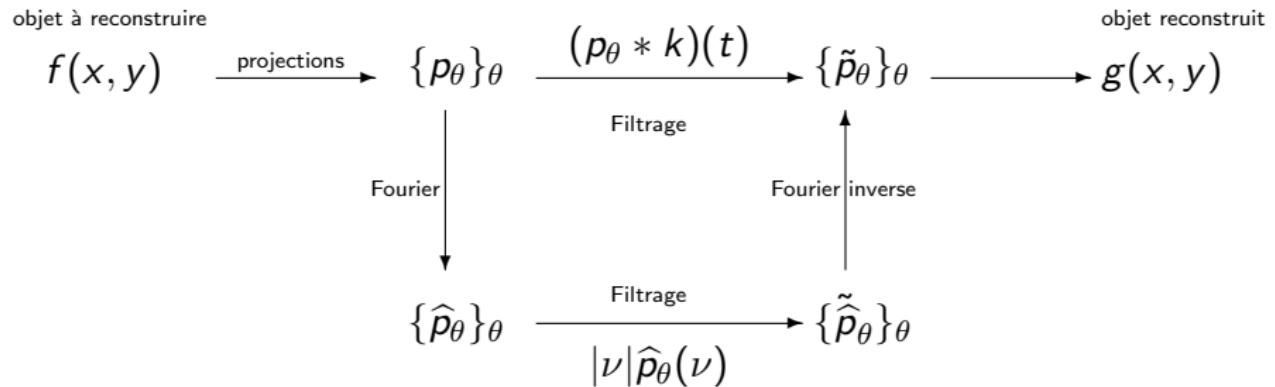


La reconstruction - Inversion analytique

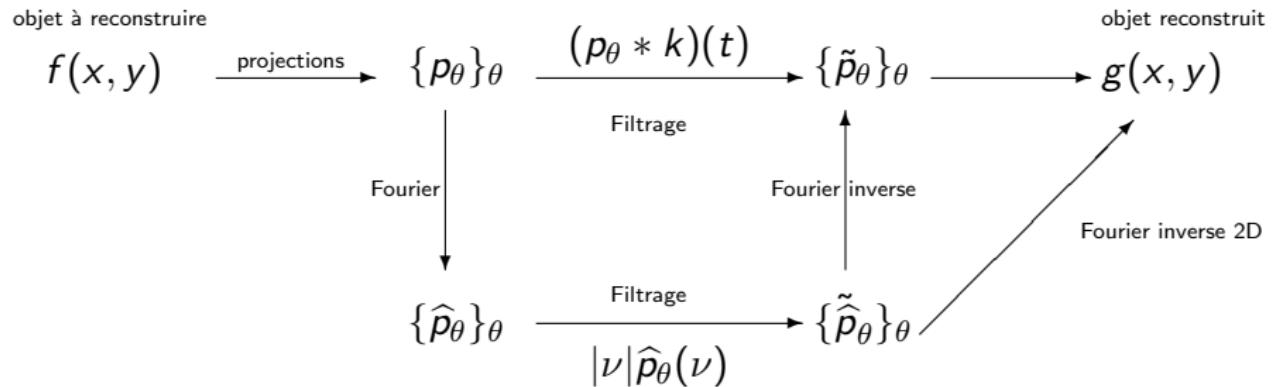
objet à reconstruire



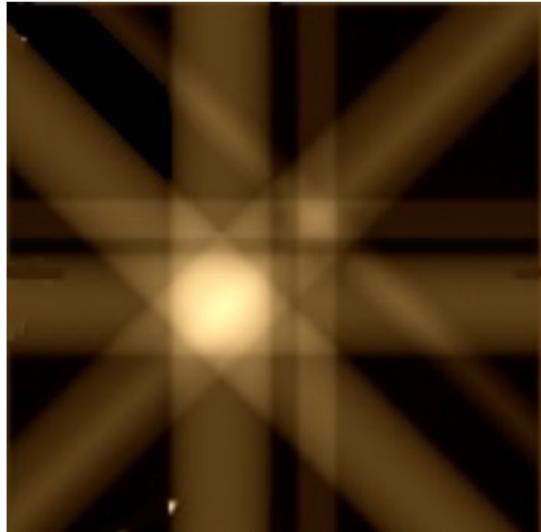
La reconstruction - Inversion analytique



La reconstruction - Inversion analytique

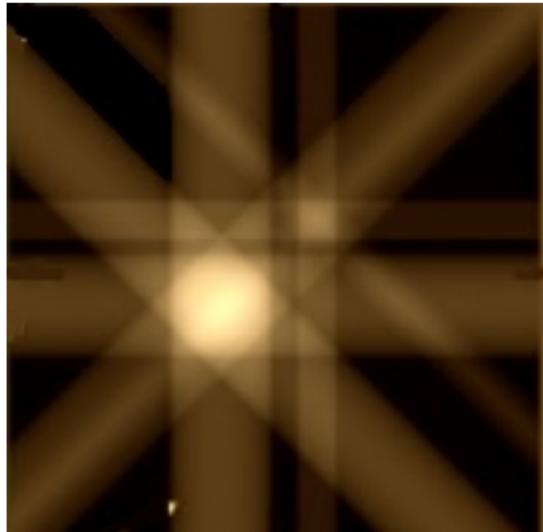


La reconstruction - Inversion analytique

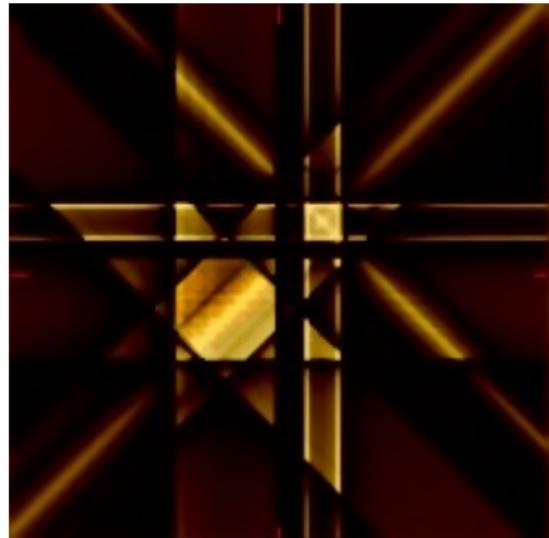


(a) Rétroprojection simple

La reconstruction - Inversion analytique



(a) Rétroprojection simple



(b) Rétroprojection filtrée

La reconstruction - Inversion discrète

- Nécessité de discréteriser les procédés pour les programmer en Python.
- Deux méthodes :
 - La transformée de Fourier discrète
 - La méthode ART ou méthode de Kaczmarz

Plan

1 Introduction

- La Transformée de Radon
- La loi de Beer-Lambert

2 La reconstruction tomographique

- Utilisation des sinogrammes
- La résolution analytique
 - La rétroprojection simple
 - La rétroprojection filtrée

3 La résolution discrète

- Une première méthode : la transformée de Fourier discrète
- Une deuxième méthode : reconstruction algébrique

4 Les résultats obtenus avec les codes Python

5 Annexe

- La transformée de Fourier
- La transformée de Fourier discrète
- Code Python : méthode par la transformée de Fourier discrète
- Code Python : méthode ART

La reconstruction - Inversion discrète

Transformée de Fourier discrète

Soit f une fonction estimée aux points (u_1, \dots, u_N) on définit la transformée de Fourier discrète (TFD) de f par la suite $(F(u_1), \dots, F(u_N))$ où :

$$\forall k \in \llbracket 1, N \rrbracket, F(u_k) = \sum_{j=1}^N f(u_j) \exp\left(\frac{-2i\pi kj}{N}\right)$$

Comme pour la transformée de Fourier on adapte la définition aux dimensions supérieures.

Theorème de la coupe centrale

Soient f la fonction caractéristique de notre objet, θ_k , un angle échantillonné et $(u_I \cos \theta_k, u_I \sin \theta_k)$ un point d'une des droites d'échantillonnage. On a :

$$P_{\theta_k}(u_I) = F(u_I \cos \theta_k, u_I \sin \theta_k)$$

Plan

1 Introduction

- La Transformée de Radon
- La loi de Beer-Lambert

2 La reconstruction tomographique

- Utilisation des sinogrammes
- La résolution analytique
 - La rétroprojection simple
 - La rétroprojection filtrée

3 La résolution discrète

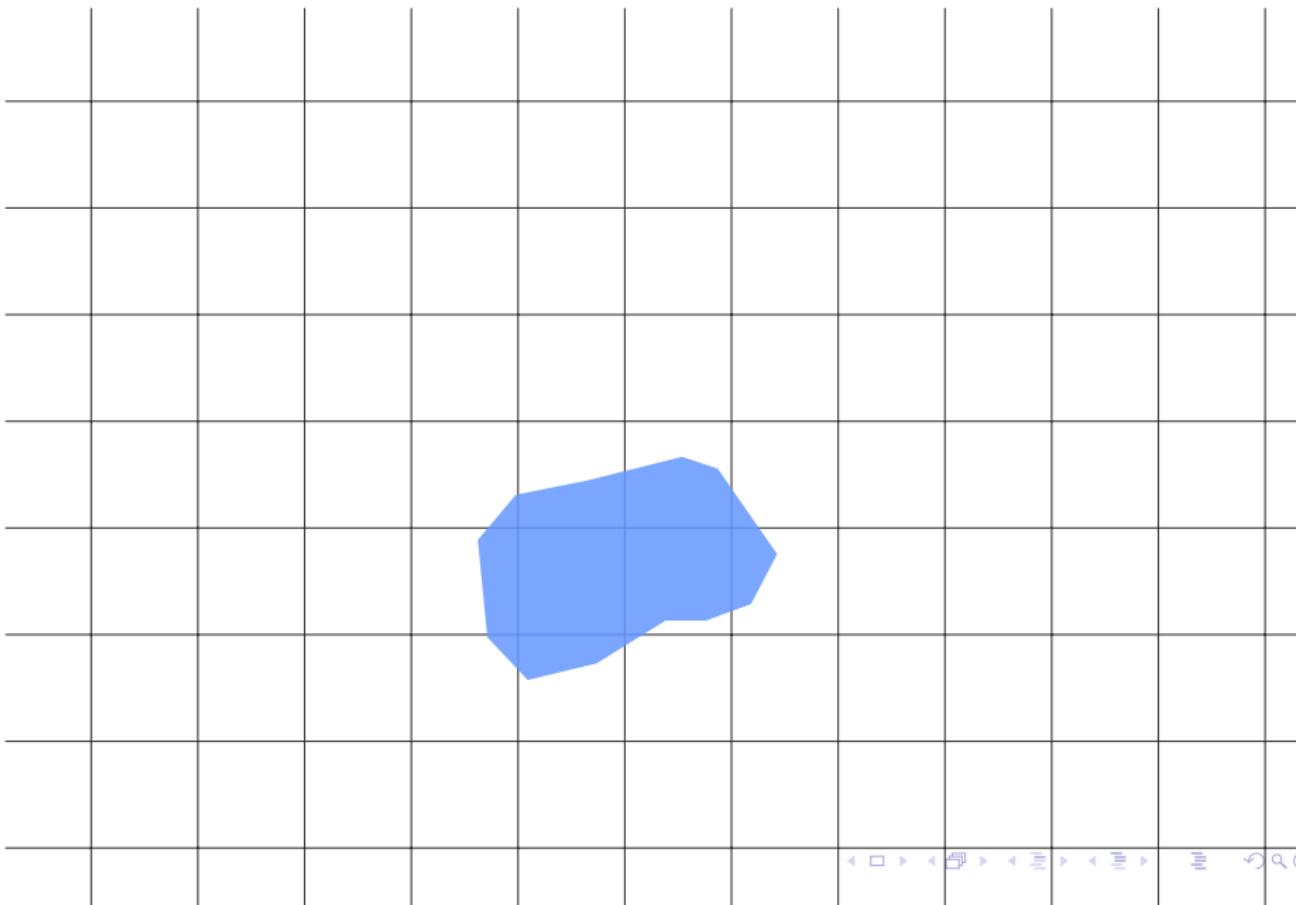
- Une première méthode : la transformée de Fourier discrète
- Une deuxième méthode : reconstruction algébrique

4 Les résultats obtenus avec les codes Python

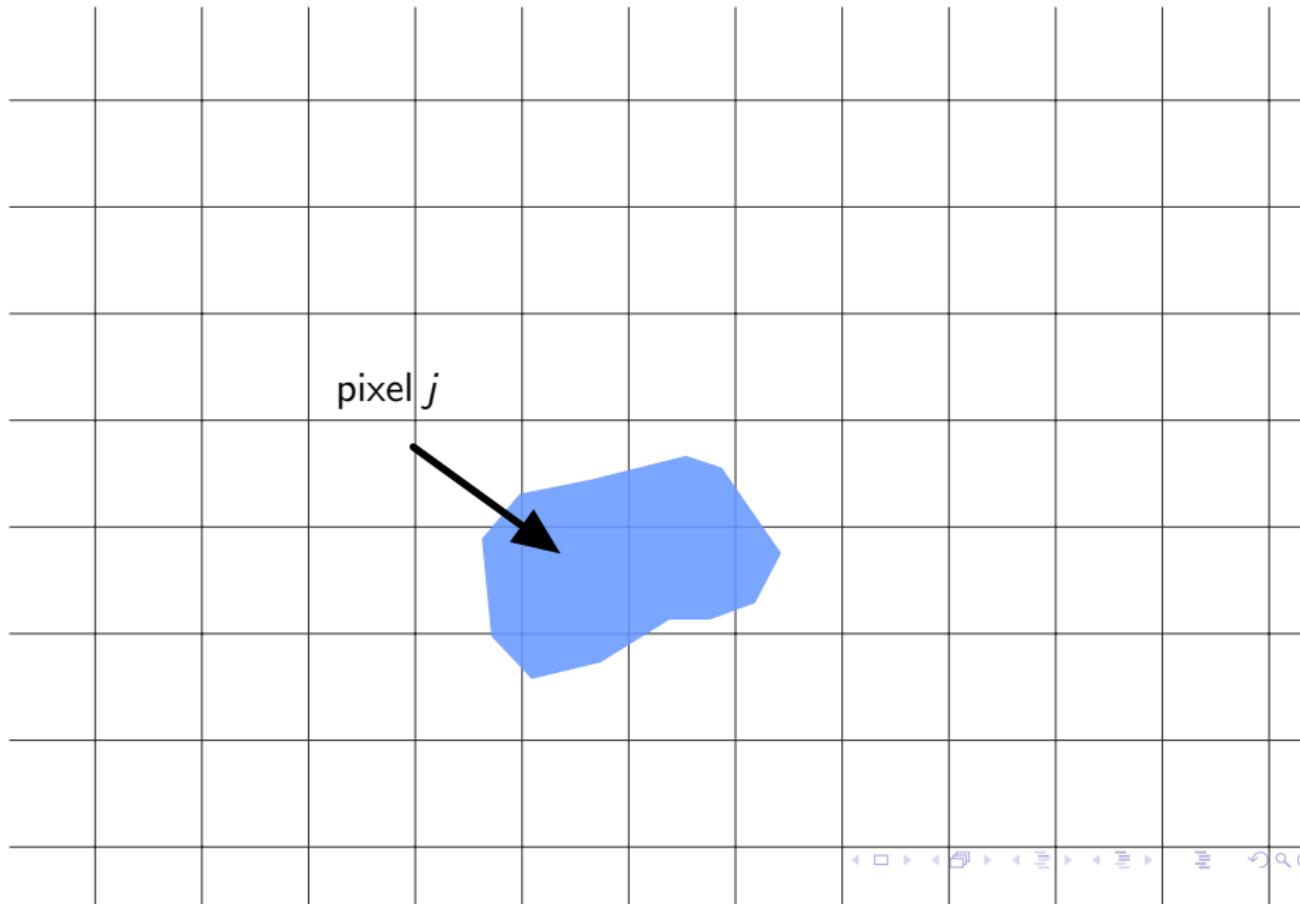
5 Annexe

- La transformée de Fourier
- La transformée de Fourier discrète
- Code Python : méthode par la transformée de Fourier discrète
- Code Python : méthode ART

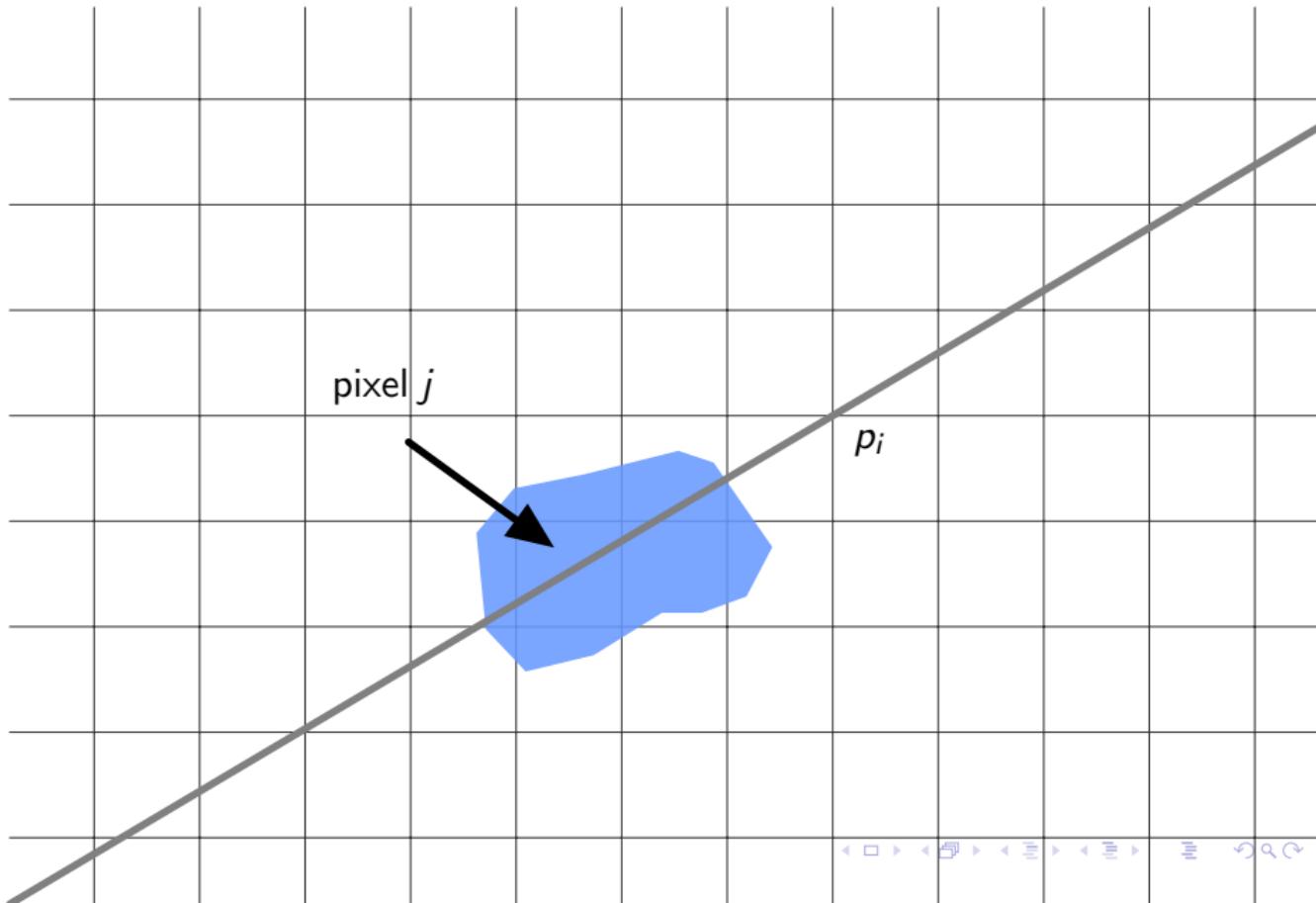
La reconstruction - Inversion discrète



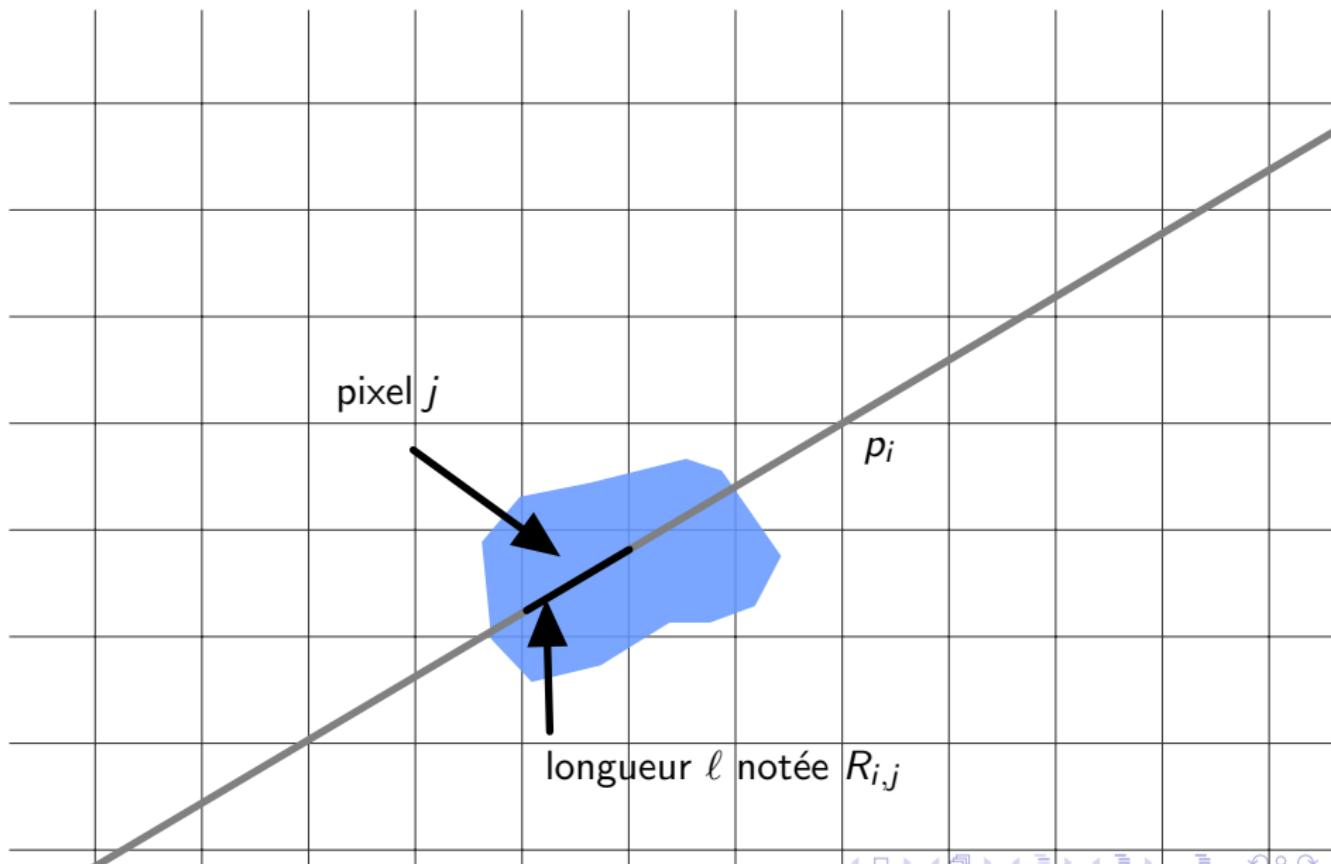
La reconstruction - Inversion discrète



La reconstruction - Inversion discrète



La reconstruction - Inversion discrète



La reconstruction - Inversion discrète

Définition des matrices

matrice de projection : $P = (p_j)_j$ qui recense les valeurs des projections

matrice de rétroposition : $R = (R_{i,j})_{i,j}$

La reconstruction - Inversion discrète

Définition des matrices

matrice de projection : $P = (p_j)_j$ qui recense les valeurs des projections

matrice de rétroposition : $R = (R_{i,j})_{i,j}$

La définition de la matrice R peut varier selon l'appareil étudié. On peut par exemple simplifier en définissant la matrice R de la sorte :

$$\forall (i,j) \in [\![1, N]\!] \times [\![1, M]\!], R_{i,j} = \begin{cases} 1 & \text{si le rayon } i \text{ passe par le pixel } j \\ 0 & \text{sinon.} \end{cases}$$

La reconstruction - Inversion discrète

Définition des matrices

matrice de projection : $P = (p_j)_j$ qui recense les valeurs des projections

matrice de rétroposition : $R = (R_{i,j})_{i,j}$

La définition de la matrice R peut varier selon l'appareil étudié. On peut par exemple simplifier en définissant la matrice R de la sorte :

$$\forall (i,j) \in [\![1, N]\!] \times [\![1, M]\!], R_{i,j} = \begin{cases} 1 & \text{si le rayon } i \text{ passe par le pixel } j \\ 0 & \text{sinon.} \end{cases}$$

Théorème

L'image F à reconstruire est solution du système matriciel $RF = P$

La reconstruction - Inversion discrète

Définition des matrices

matrice de projection : $P = (p_j)_j$ qui recense les valeurs des projections

matrice de rétroposition : $R = (R_{i,j})_{i,j}$

La définition de la matrice R peut varier selon l'appareil étudié. On peut par exemple simplifier en définissant la matrice R de la sorte :

$$\forall (i,j) \in [\![1, N]\!] \times [\![1, M]\!], R_{i,j} = \begin{cases} 1 & \text{si le rayon } i \text{ passe par le pixel } j \\ 0 & \text{sinon.} \end{cases}$$

Théorème

L'image F à reconstruire est solution du système matriciel $RF = P$

En effet, par définition de la matrice de projection :

$$\forall i \in [\![1, N]\!], p_i = \sum_{j=1}^M R_{i,j} f(j)$$

La reconstruction - Inversion discrète

Reformulation du problème

F est solution du système si et seulement si $F \in \bigcap_{i=1}^N \mathcal{H}_i$ où $\mathcal{H}_i = p_i + \{{}^t L_i\}^\perp$

On peut résoudre par la méthode dite de Kaczmarz :

- on choisit $F_0 \in \mathbb{R}^M$

La reconstruction - Inversion discrète

Reformulation du problème

F est solution du système si et seulement si $F \in \bigcap_{i=1}^N \mathcal{H}_i$ où $\mathcal{H}_i = p_i + \{{}^t L_i\}^\perp$

On peut résoudre par la méthode dite de Kaczmarz :

- on choisit $F_0 \in \mathbb{R}^M$
- on projette orthogonalement F_0 sur \mathcal{H}_1 pour obtenir F_1

La reconstruction - Inversion discrète

Reformulation du problème

F est solution du système si et seulement si $F \in \bigcap_{i=1}^N \mathcal{H}_i$ où $\mathcal{H}_i = p_i + \{{}^t L_i\}^\perp$

On peut résoudre par la méthode dite de Kaczmarz :

- on choisit $F_0 \in \mathbb{R}^M$
- on projette orthogonalement F_0 sur \mathcal{H}_1 pour obtenir F_1
- on projette orthogonalement F_1 sur \mathcal{H}_2 pour obtenir F_2
- ...

La reconstruction - Inversion discrète

Reformulation du problème

F est solution du système si et seulement si $F \in \bigcap_{i=1}^N \mathcal{H}_i$ où $\mathcal{H}_i = p_i + \{{}^t L_i\}^\perp$

On peut résoudre par la méthode dite de Kaczmarz :

- on choisit $F_0 \in \mathbb{R}^M$
- on projette orthogonalement F_0 sur \mathcal{H}_1 pour obtenir F_1
- on projette orthogonalement F_1 sur \mathcal{H}_2 pour obtenir F_2
- ...
- on projette orthogonalement F_{N-1} sur \mathcal{H}_N pour obtenir F_N

La reconstruction - Inversion discrète

Reformulation du problème

F est solution du système si et seulement si $F \in \bigcap_{i=1}^N \mathcal{H}_i$ où $\mathcal{H}_i = p_i + \{{}^t L_i\}^\perp$

On peut résoudre par la méthode dite de Kaczmarz :

- on choisit $F_0 \in \mathbb{R}^M$
- on projette orthogonalement F_0 sur \mathcal{H}_1 pour obtenir F_1
- on projette orthogonalement F_1 sur \mathcal{H}_2 pour obtenir F_2
- ...
- on projette orthogonalement F_{N-1} sur \mathcal{H}_N pour obtenir F_N
- puis on recommence à partir du vecteur F_N ...

La reconstruction - Inversion discrète

Expression de la suite

La suite est alors définie par récurrence par la formule :

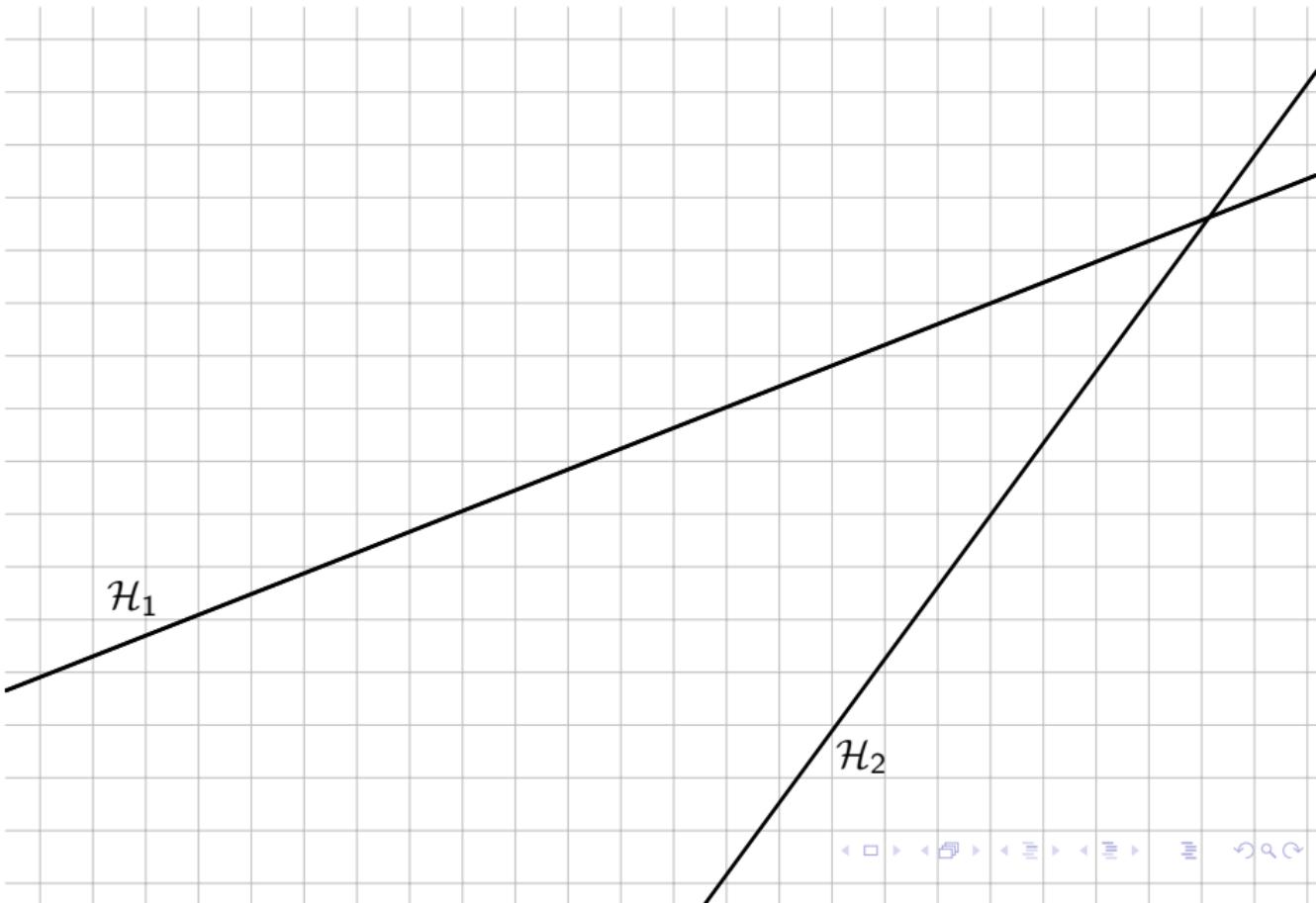
$$\left\{ \begin{array}{l} F_0 \in M_{M,1}(\mathbb{R}) \\ \forall n \in \mathbb{N}^*, F_{n+1} = F_n - \frac{\langle F_n - q_{r(n)+1}, N_{r(n)+1} \rangle}{\|N_{r(n)+1}\|^2} N_{r(n)+1} \end{array} \right.$$

où $r(n)$ est le reste de la division euclidienne de n par N .

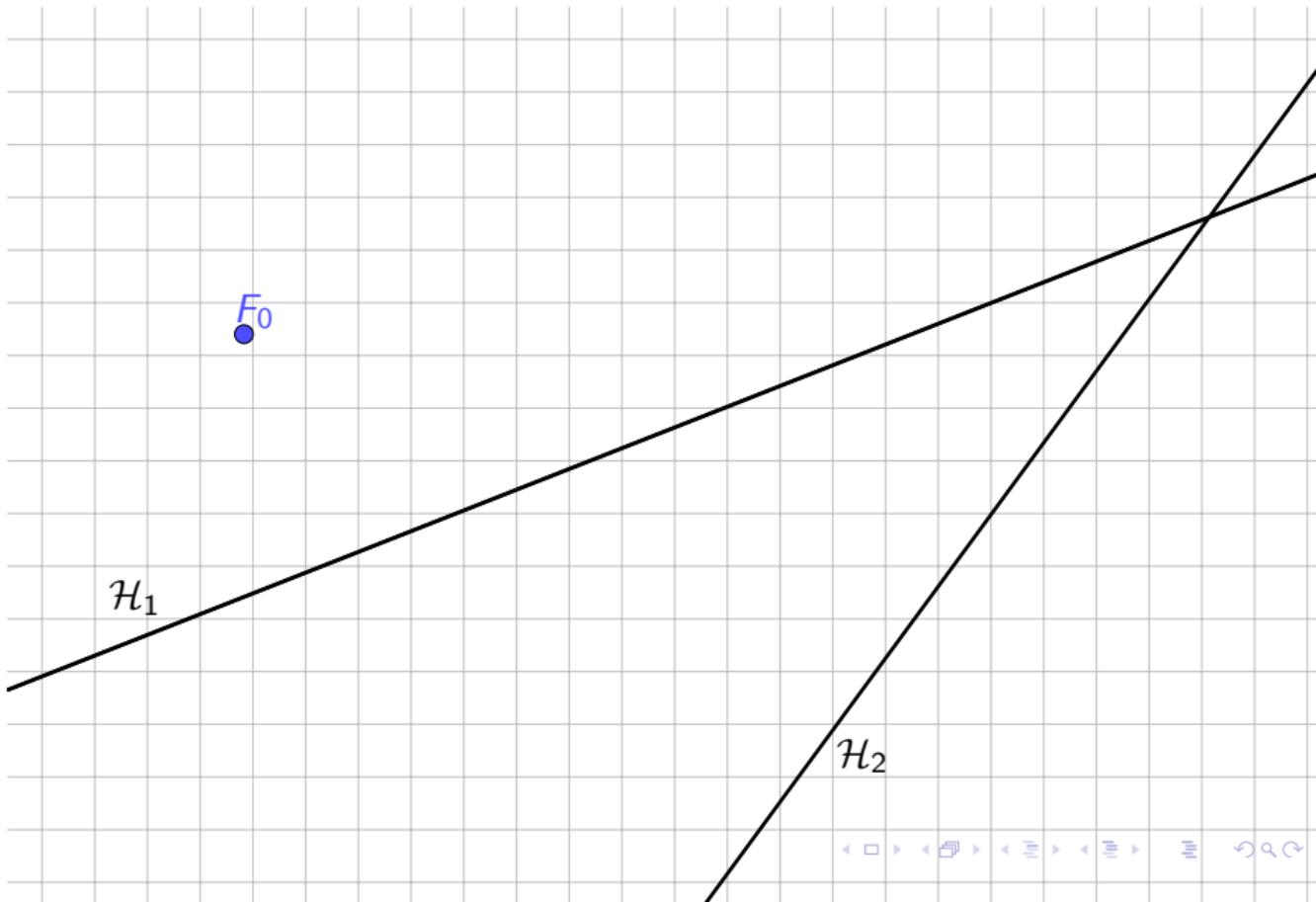
cette expression découle de l'expression du projeté orthogonal d'un vecteur x sur un hyperplan $H = \{a\}^\perp$:

$$p_H(x) = x - \frac{\langle x, a \rangle}{\|a\|^2} a$$

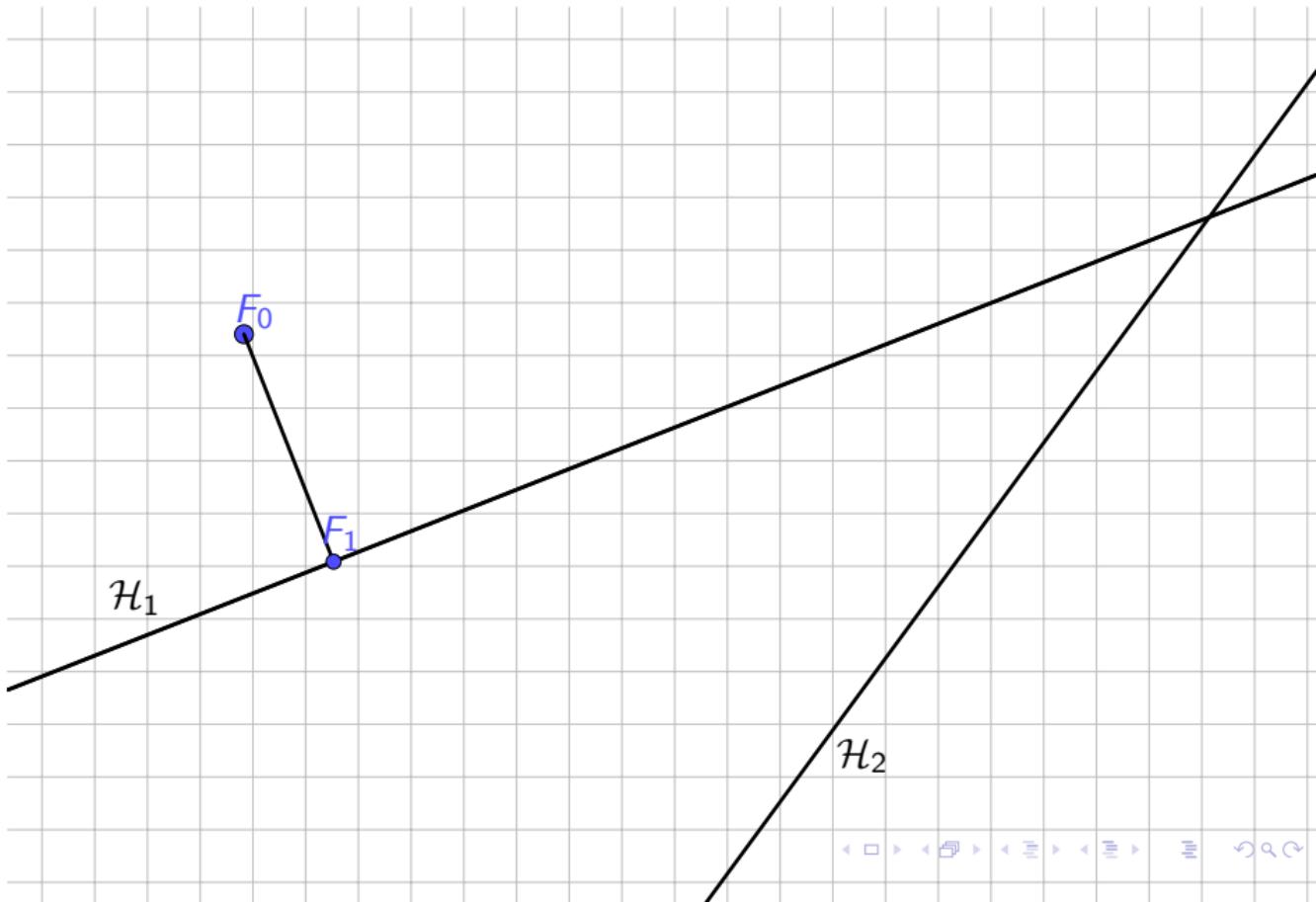
La reconstruction - Inversion discrète



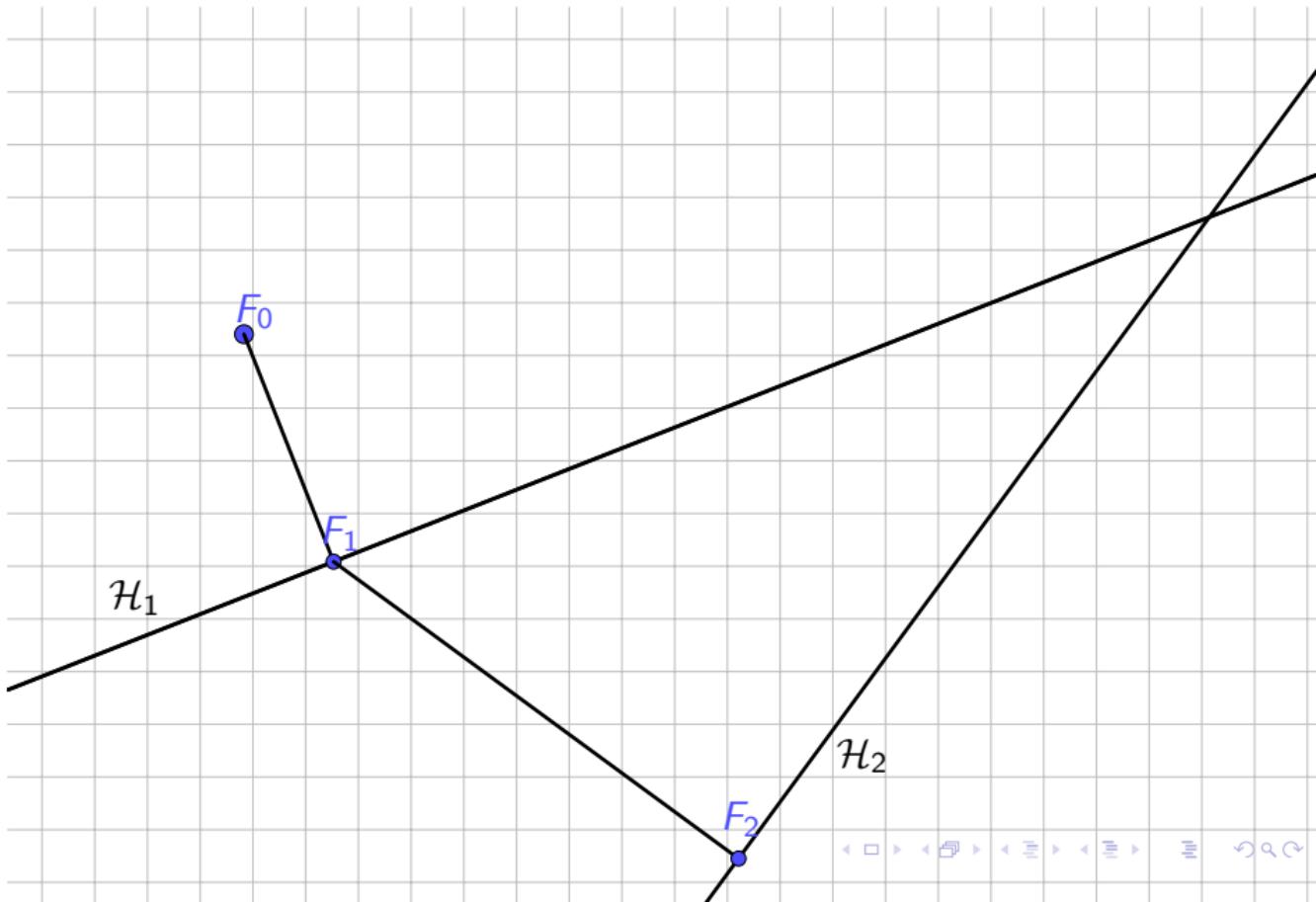
La reconstruction - Inversion discrète



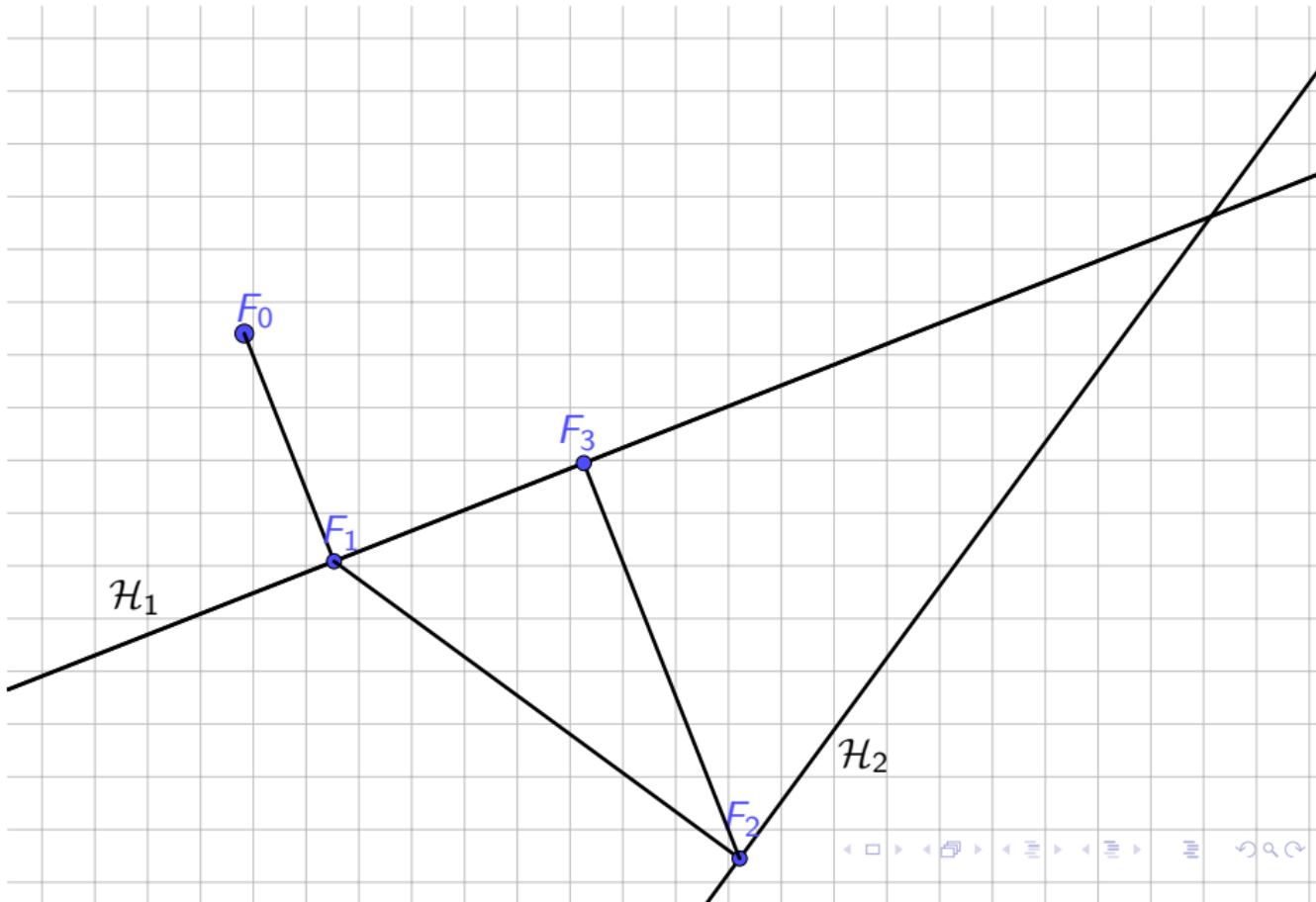
La reconstruction - Inversion discrète



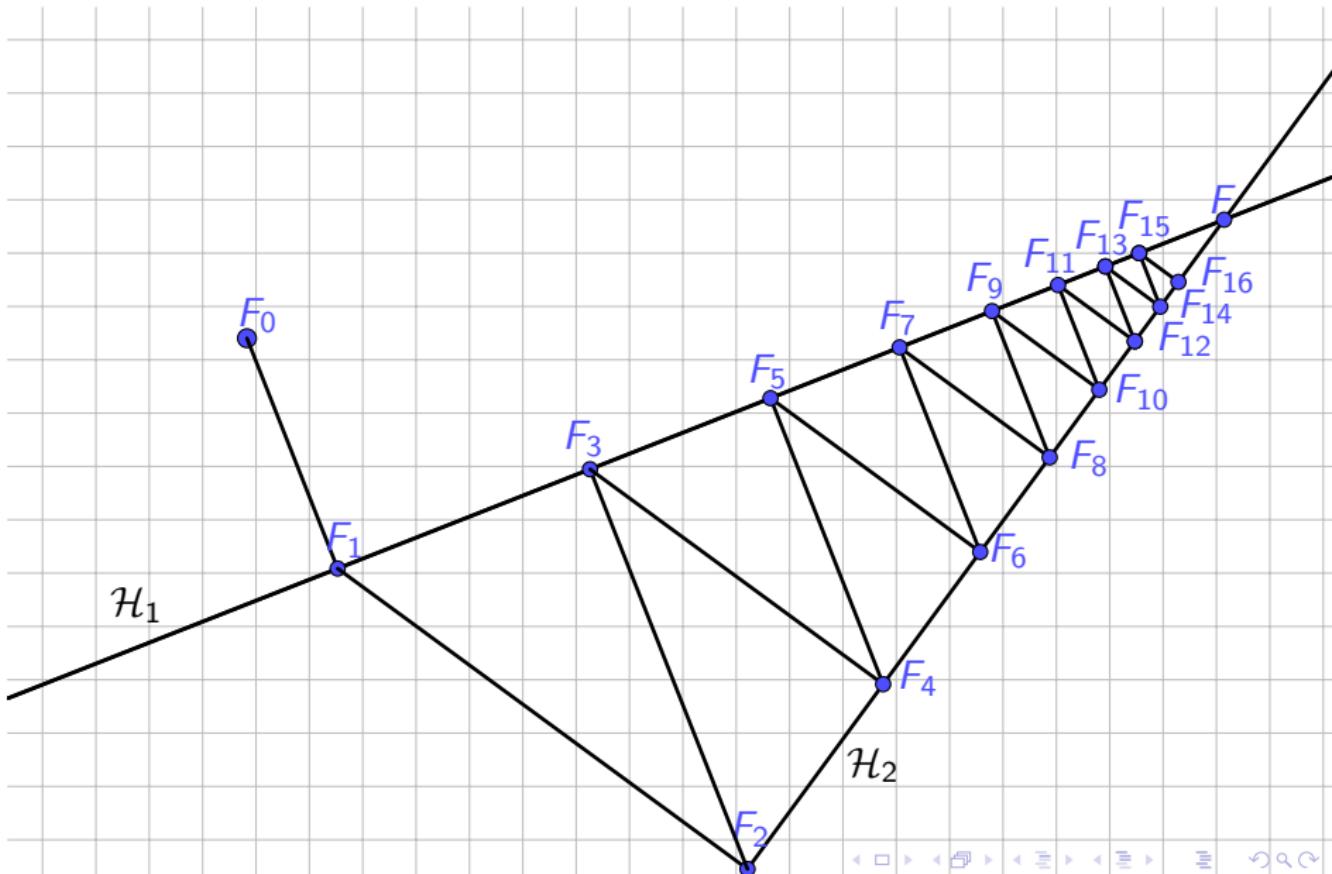
La reconstruction - Inversion discrète



La reconstruction - Inversion discrète



La reconstruction - Inversion discrète



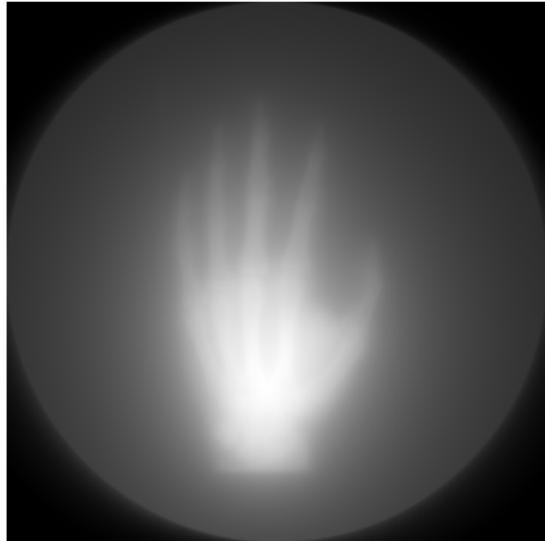
Résultats - première méthode

Reconstruction de la main (155 400 pixels) :



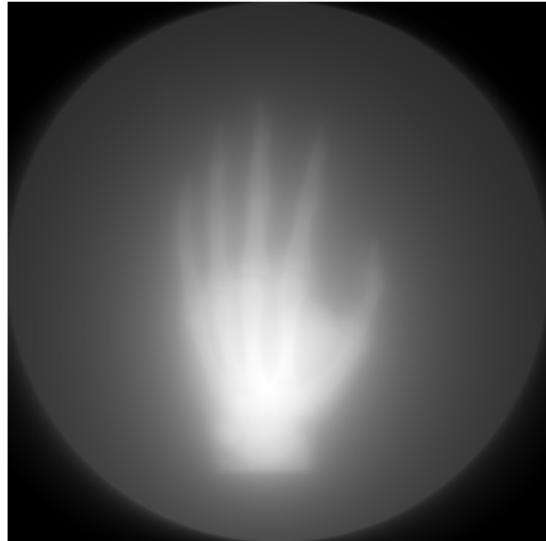
Figure: Radiographie originale

Résultats - première méthode



(a) Sans transformée de Fourier

Résultats - première méthode



(a) Sans transformée de Fourier



(b) Avec transformée de Fourier

Résultats - première méthode

Reconstruction de la mâchoire (135 000 pixels) :

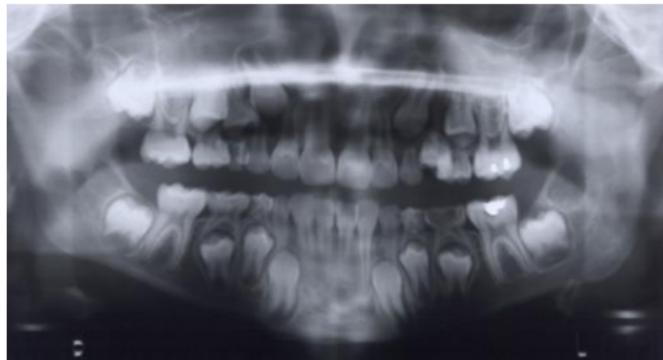
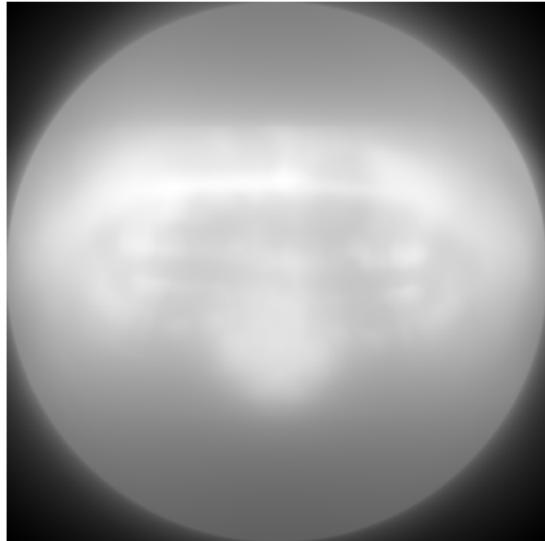


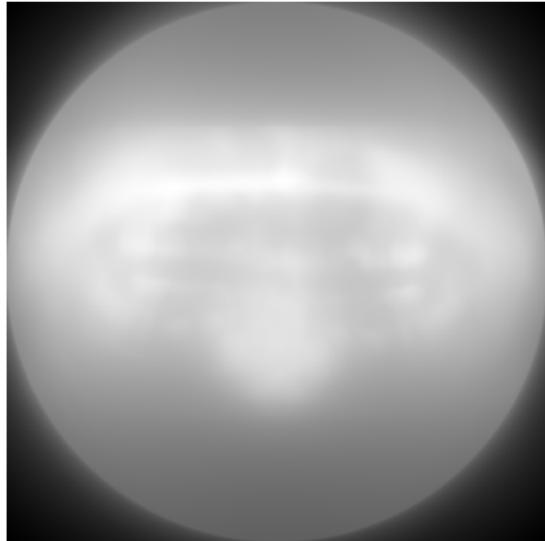
Figure: Radiographie originale

Résultats - première méthode

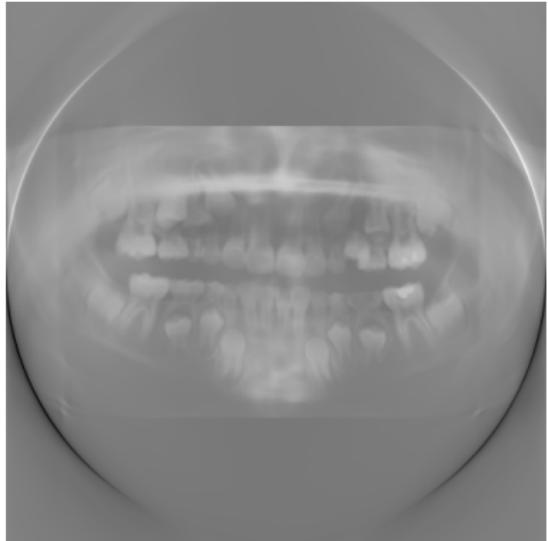


(a) Sans transformée de Fourier

Résultats - première méthode



(a) Sans transformée de Fourier



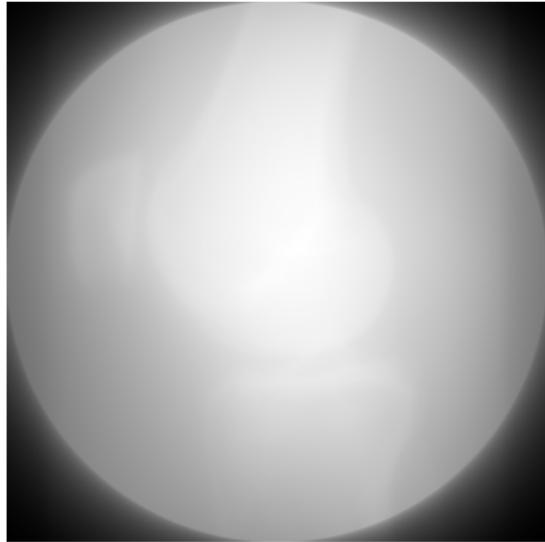
(b) Avec transformée de Fourier

Résultats - première méthode

Reconstruction du genou (1 675 500 pixels) :

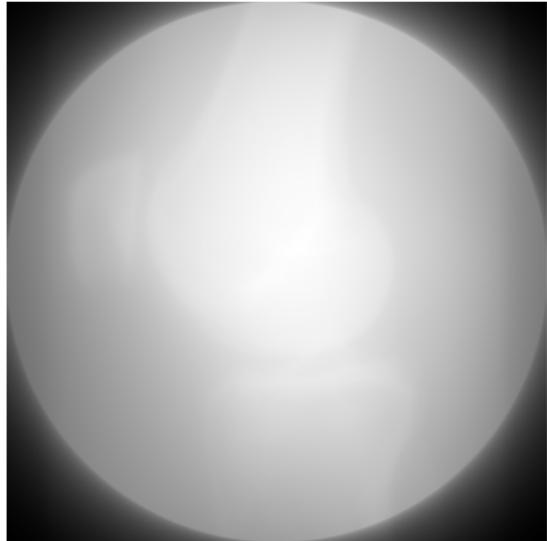


Résultats - première méthode



(a) Sans transformée de Fourier

Résultats - première méthode



(a) Sans transformée de Fourier



(b) Avec transformée de Fourier

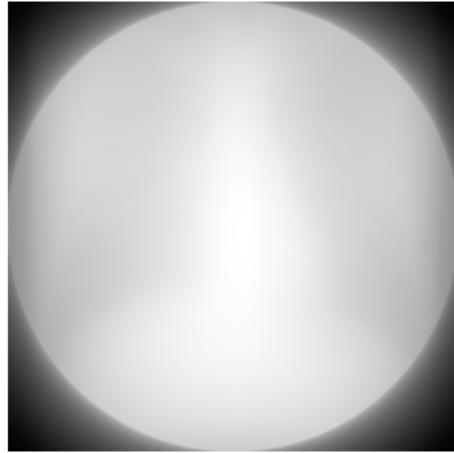
Résultats - première méthode

Reconstruction du thorax (5 095 242 pixels) :



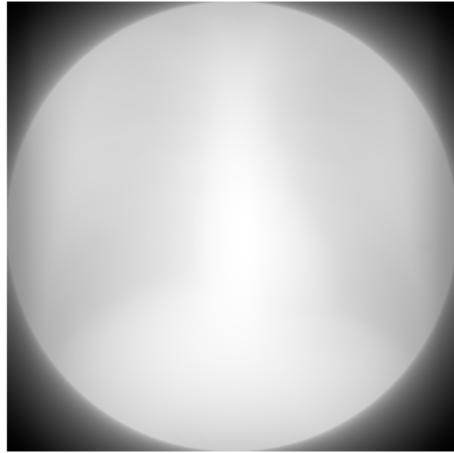
Figure: Radiographie originale

Résultats - première méthode

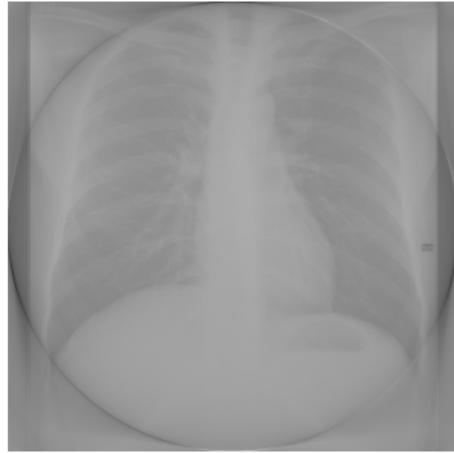


(a) Sans transformée de Fourier

Résultats - première méthode

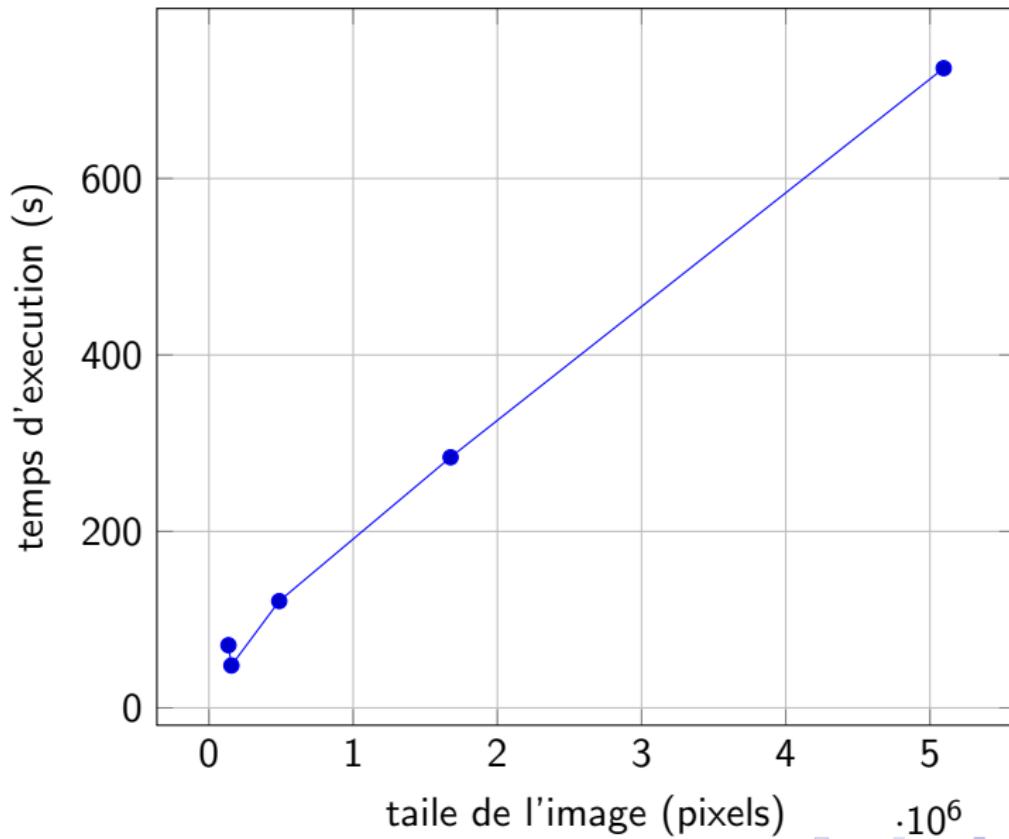


(a) Sans transformée de Fourier



(b) Avec transformée de Fourier

Résultats - première méthode



Résultats - deuxième méthode

Un premier essai avec la matrice R simple :

$$\forall (i, j) \in [\![1, N]\!] \times [\![1, M]\!], R_{i,j} = \begin{cases} 1 & \text{si le rayon } i \text{ passe par le pixel } j \\ 0 & \text{sinon.} \end{cases}$$

Résultats - deuxième méthode

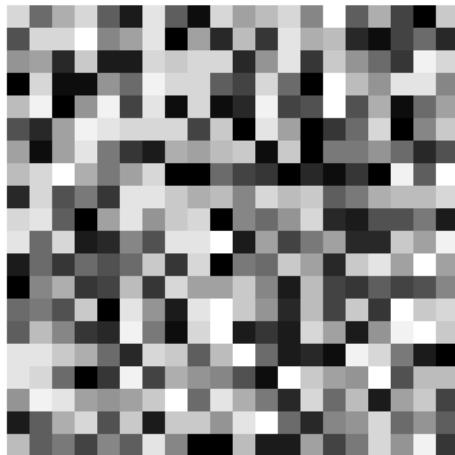
Un premier essai avec la matrice R simple :

$$\forall (i, j) \in [\![1, N]\!] \times [\![1, M]\!], R_{i,j} = \begin{cases} 1 & \text{si le rayon } i \text{ passe par le pixel } j \\ 0 & \text{sinon.} \end{cases}$$

- images choisies : matrices aléatoires de taille 20×20
- 1000 projections
- itérations : de 3000 à 10000000

Résultats - deuxième méthode

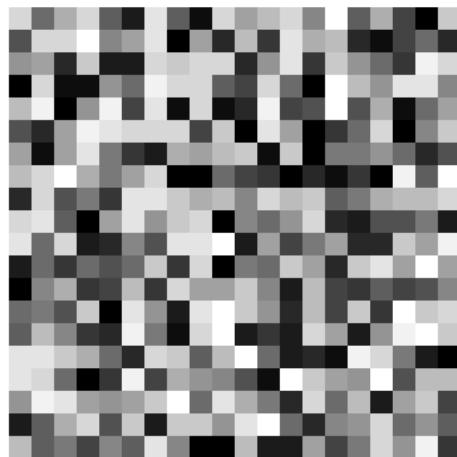
Reconstruction avec 3000 itérations



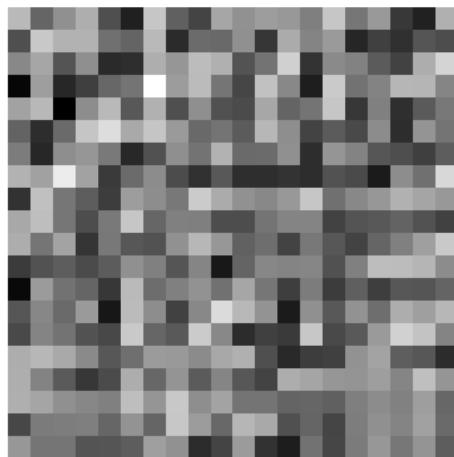
(a) Matrice originale

Résultats - deuxième méthode

Reconstruction avec 3000 itérations



(a) Matrice originale



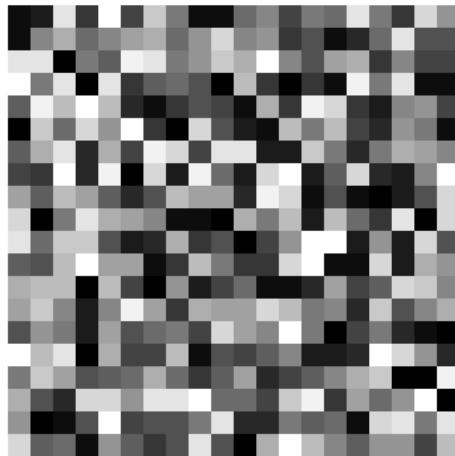
(b) Matrice reconstruite

Erreur maximale : 7.6729633974723335

Erreurs cumulées : 679.4563049009196

Résultats - deuxième méthode

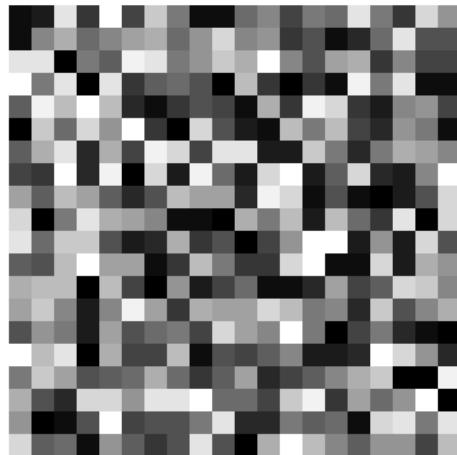
Reconstruction avec 100000 itérations



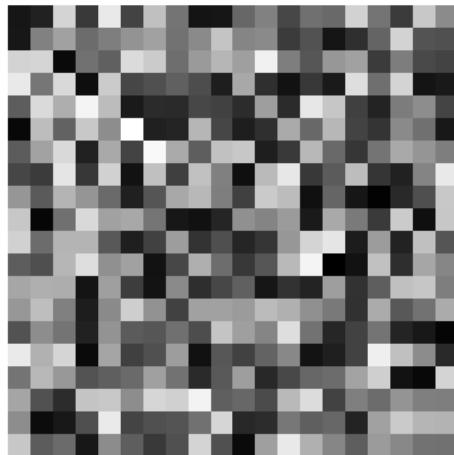
(a) Matrice originale

Résultats - deuxième méthode

Reconstruction avec 100000 itérations



(a) Matrice originale



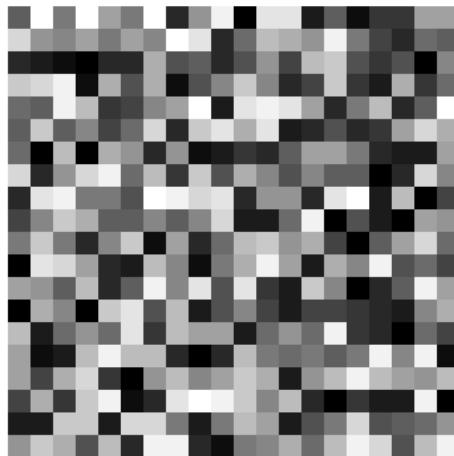
(b) Matrice reconstruite

Erreur maximale : 2.709169928559728

Erreurs cumulées : 191.0245806067057

Résultats - deuxième méthode

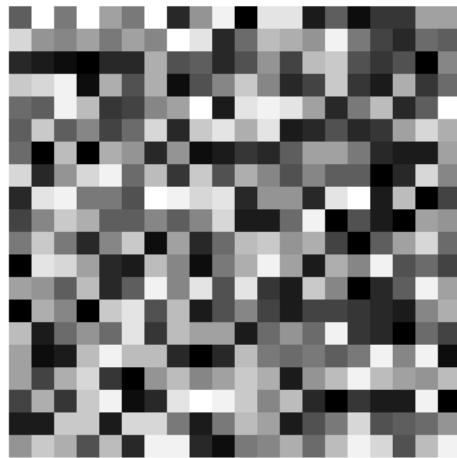
Reconstruction avec 10000000 itérations



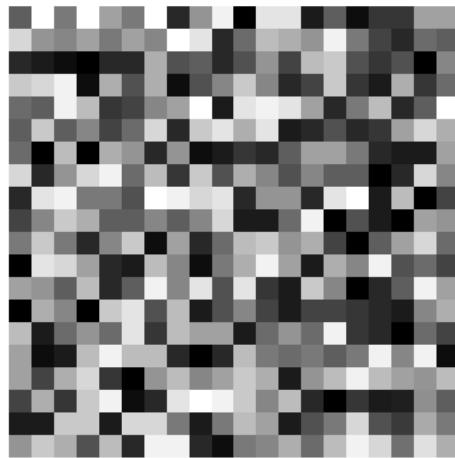
(a) Matrice originale

Résultats - deuxième méthode

Reconstruction avec 10000000 itérations



(a) Matrice originale

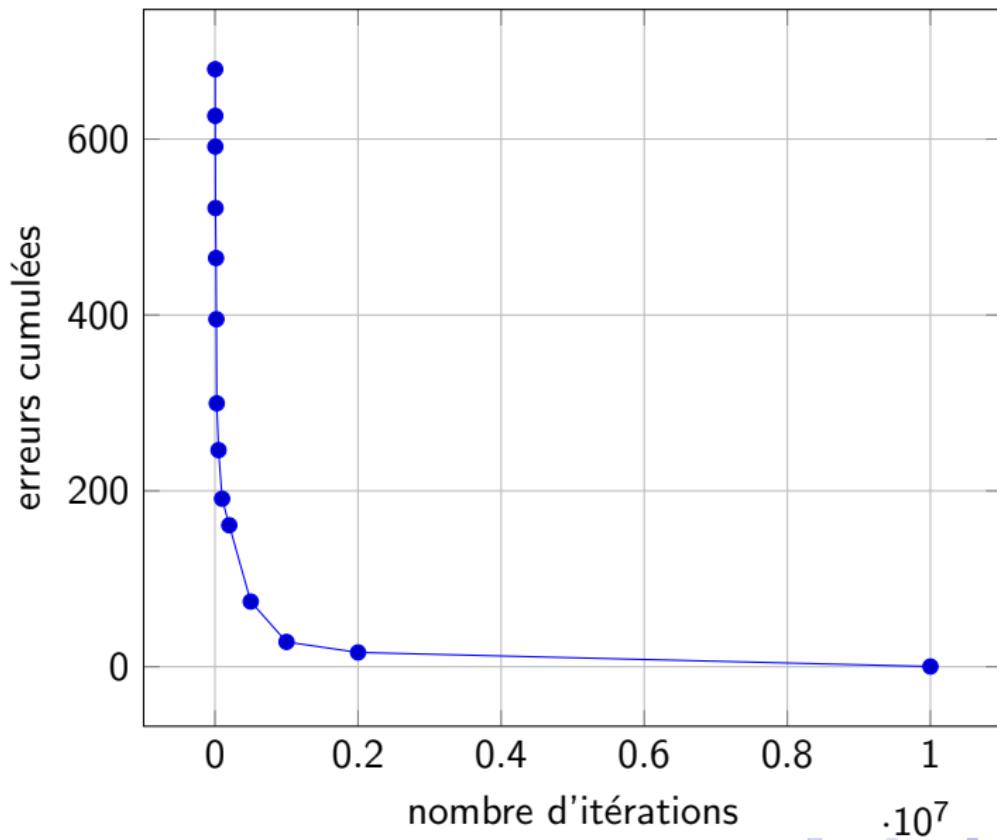


(b) Matrice reconstruite

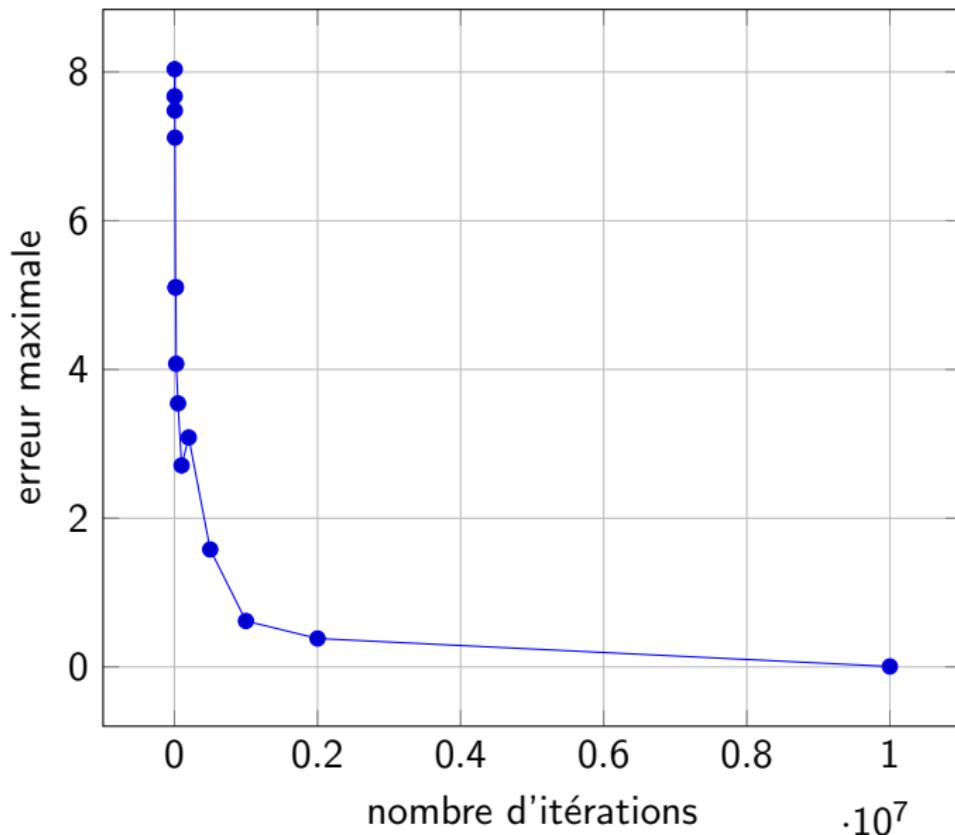
Erreur maximale : 0.006320163485931118

Erreurs cumulées : 0.2761238448059684

Résultats - deuxième méthode



Résultats - deuxième méthode



Plan

1 Introduction

- La Transformée de Radon
- La loi de Beer-Lambert

2 La reconstruction tomographique

- Utilisation des sinogrammes
- La résolution analytique
 - La rétroprojection simple
 - La rétroprojection filtrée

3 La résolution discrète

- Une première méthode : la transformée de Fourier discrète
- Une deuxième méthode : reconstruction algébrique

4 Les résultats obtenus avec les codes Python

5 Annexe

- La transformée de Fourier
- La transformée de Fourier discrète
- Code Python : méthode par la transformée de Fourier discrète
- Code Python : méthode ART

Transformée de Fourier

Définitions

En dimension 1 : Soit $f : \mathbb{R} \mapsto \mathbb{R}$ une fonction intégrable on définit la transformée de Fourier par :

$$\forall \nu \in \mathbb{R}, \widehat{f}(\nu) = \int_{\mathbb{R}} f(t) e^{-2i\pi\nu t} dt$$

En dimension 2 : Soit $f : \mathbb{R}^2 \mapsto \mathbb{R}$ une fonction intégrable on définit la transformée de Fourier par :

$$\forall (u, v) \in \mathbb{R}^2, \widehat{f}(u, v) = \int_{\mathbb{R}^2} f(x, y) e^{-2i\pi(ux+vy)} dx dy$$

Transformée de Fourier

Théorème d'inversion de Fourier

Sous certaines hypothèses supposées vérifiées dans le cadre de notre étude on peut accéder à f par inversion de sa transformée de Fourier.

En dimension 1 :

$$\forall t \in \mathbb{R}, f(t) = \int_{\mathbb{R}} \hat{f}(\nu) e^{2i\pi\nu t} d\nu$$

En dimension 2 :

$$\forall (x, y) \in \mathbb{R}^2, f(x, y) = \int_{\mathbb{R}^2} \hat{f}(u, v) e^{2i\pi(ux+vy)} du dv$$

Il est par exemple suffisant que f soit dans l'espace de Schwartz $\mathcal{S}(\mathbb{R}^d)$ ($d \in \{1, 2\}$) où :

$$\mathcal{S}(\mathbb{R}) = \{f \in \mathcal{C}^\infty : \forall (n, m) \in \mathbb{N}, x \mapsto x^n f^{(m)}(x) \text{ bornée}\}$$

$$\mathcal{S}(\mathbb{R}^2) = \{f \in \mathcal{C}^\infty : \forall (p, q, n, m) \in \mathbb{N}^4, (x, y) \mapsto x^p y^q \partial_x^n \partial_y^m f(x, y) \text{ bornée}\}$$

Plan

1 Introduction

- La Transformée de Radon
- La loi de Beer-Lambert

2 La reconstruction tomographique

- Utilisation des sinogrammes
- La résolution analytique
 - La rétroprojection simple
 - La rétroprojection filtrée

3 La résolution discrète

- Une première méthode : la transformée de Fourier discrète
- Une deuxième méthode : reconstruction algébrique

4 Les résultats obtenus avec les codes Python

5 Annexe

- La transformée de Fourier
- La transformée de Fourier discrète
- Code Python : méthode par la transformée de Fourier discrète
- Code Python : méthode ART

Transformée de Fourier discrète

Transformée de Fourier discrète

Soit f une fonction estimée aux points (u_1, \dots, u_N) on définit la transformée de Fourier discrète (TFD) de f par la suite $(F(u_1), \dots, F(u_N))$ où :

$$\forall k \in \llbracket 1, N \rrbracket, F(u_k) = \sum_{l=1}^N f(u_l) \exp\left(\frac{-2i\pi kl}{N}\right)$$

Expression matricielle

$$\begin{pmatrix} F(u_1) \\ \vdots \\ F(u_N) \end{pmatrix} = \Omega \begin{pmatrix} f(u_1) \\ \vdots \\ f(u_N) \end{pmatrix} \text{ avec } \Omega = (\omega^{kl})_{(k,l)} \text{ et } \omega = e^{\frac{-2i\pi}{N}}$$

Transformée de Fourier discrète

Spécificité de la matrice Ω

La matrice $\frac{\Omega}{\sqrt{N}}$ est dite unitaire.

Soient $k, l \in \llbracket 1, N \rrbracket$,

$$[\Omega^* \Omega]_{k,l} = \sum_{d=1}^N [{}^t \bar{\Omega}]_{k,d} [\Omega]_{d,l} = \sum_{d=1}^N \omega^{-kd} \omega^{dl} = \sum_{d=1}^N \omega^{d(l-k)} = \delta_{k,l} N$$

Donc :

$$\left(\frac{\Omega}{\sqrt{N}} \right)^* \frac{\Omega}{\sqrt{N}} = I_N$$

Conséquences : Ω est inversible et s'inverse facilement

Transformée de Fourier discrète

Transformée de Fourier discrète - dimension 2

Soit f une fonction estimée aux points $(u_1, \dots, u_N, v_1, \dots, v_M)$ on définit la transformée de Fourier discrète 2D de f par la suite double $(F(u_1, v_1), F(u_1, v_2), \dots, F(u_N, v_M))$ où :

$$\forall k, l \in [\![1, N]\!] \times [\![1, M]\!], F(u_k, u_l) = \sum_{n=1}^N \sum_{m=1}^M f(u_n, u_m) e^{\frac{-2i\pi kn}{N}} e^{\frac{-2i\pi lm}{M}}$$

Plan

1 Introduction

- La Transformée de Radon
- La loi de Beer-Lambert

2 La reconstruction tomographique

- Utilisation des sinogrammes
- La résolution analytique
 - La rétroprojection simple
 - La rétroprojection filtrée

3 La résolution discrète

- Une première méthode : la transformée de Fourier discrète
- Une deuxième méthode : reconstruction algébrique

4 Les résultats obtenus avec les codes Python

5 Annexe

- La transformée de Fourier
- La transformée de Fourier discrète
- Code Python : méthode par la transformée de Fourier discrète
- Code Python : méthode ART

Annexe - Code méthode par la transformée de Fourier discrète I

```
1 import imageio
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from PIL import Image
5 from math import cos, sin, pi, sqrt
6 import scipy.fftpack as TF
7 from skimage.transform import rotate
8
9
10 def imread(filename, greyscale=True):
11     """ Transformation d'une image en tableau """
12     if greyscale:
13         pil_im = Image.open(filename).convert('L')    #
14         converti en nuance de gris
15     else:
16         pil_im = Image.open(filename)
17     return np.array(pil_im)
```

Annexe - Code méthode par la transformée de Fourier discrète II

```
17
18
19 def taille(im):
20     return img.shape
21
22
23 def affiche_image(im):
24     """ Affiche une tableau numpy """
25     plt.imshow(im)
26     plt.show()
27
28
29 def saveAsPNG(im, name):
30     """ Sauve un tableau numpy en png sous le nom 'name' """
31     imageio.imwrite(name + '.png', im)
32
33
```

Annexe - Code méthode par la transformée de Fourier discrète III

```
34 # Importation de l'image
35 img = imread('original.png')
36
37
38 # %% Quelques fonctions qui vont servir :
39
40 def dansImage(im, i, j):
41     """ verifie qu'un pixel est dans l'image """
42     if i >= 0 and i < x and j >= 0 and j < y:
43         return True
44     else:
45         return False
46
47
48 def color(im, i, j, c):
49     """modifie la couleur de certains pixels de l'image"""
50     if dansImage(im, i, j):
```

Annexe - Code méthode par la transformée de Fourier discrète IV

```
51     im[i, j] = c
52
53
54 # %% Première méthode (mauvais résultats)
55
56 def proj_graph(im, m, p, horizontale=False):
57     """ Modifie la couleur des pixels sur le long d'une
58     droite de coef directeur m et d'ordonnée à l'origine p
59     """
60
61     if horizontale:
62         for i in range(0, x):
63             color(img, i, p, 255)
64             horizontale = False
65     else:
66         for i in range(-d1, d1):
67             j = round(m * i + p)
68             color(img, i, j, 255)
```

Annexe - Code méthode par la transformée de Fourier discrète V

```
66 plt.figure(dpi=300)
67 plt.imshow(img)
68 plt.show()
69
70
71 def proj(im, m, p, horizontale=False):
72     """ Projection selon la droite de coef directeur m et d'  
ordonnee a l'origine p"""
73     tot = 0
74     if horizontale:
75         for i in range(0, x):
76             tot += im[i, p]
77     else:
78         # on somme les valeurs des pixels qui sont sur la
79         # droite
80         for i in range(-d1, d1):
81             j = round(m * i + p)
```

Annexe - Code méthode par la transformée de Fourier discrète VI

```
81         if dansImage(im, i, j):
82             tot += im[i, j]
83     return tot
84
85
86 def coupe(img, m, pas=50, horizontale=False):
87     """ Fait une projection selon une direction """
88     lst = []
89     a = round(-m * x)
90     p = round((-a + x) / pas)
91     # met dans une liste les coupes d'une même direction
92     for k in range(a, x, p):
93         lst.append(proj(img, m, k, horizontale))
94     while len(lst) < pas + 15: # 15 a ajuster pour qu'il n'
95         y ai pas de dépassement du tableau
96         lst.append(0)
97     return lst
```

Annexe - Code méthode par la transformée de Fourier discrète VII

```
97
98
99 def coupe_graph(im, m, pas=50):
100     """ Affiche les droites de projections pour une
101     direction """
102     a = round(-m * x)
103     for k in np.linspace(a, x, pas):
104         for i in range(-d1, d1):
105             j = round(m * i + k)
106             color(im, i, j, 255)
107     plt.figure(dpi=300)
108     plt.imshow(img)
109     plt.show()
110
111 def sinogramme(im, pas=10):
112     """ Retourne le sinogramme """
```

Annexe - Code méthode par la transformée de Fourier discrète VIII

```
13     l = np.linspace(10, 0, pas)
14     projections = []
15     for k in l:
16         projections.append(coupe(img, k))
17     return np.vstack(projections)
18
19
20 # %% Deuxieme methode avec des coordonnees polaires
21
22 def coupe_graph_polaire(img, theta, s):
23     """ Affiche une droite de projection """
24     for k in range(-int(1.4 * s), int(1.4 * s)):
25         i, j = round(s * cos(theta) - k * sin(theta)), round(
26             (s * sin(theta) + k * cos(theta)))
27         color(img, i, j, 255)
28
plt.figure(dpi=300)
```

Annexe - Code méthode par la transformée de Fourier discrète IX

```
29     plt.imshow(img)
30     plt.show()
31
32
33 def coupe_polaire(im, theta, s):
34     """ Projection selon une droite de parametre theta, s
35     """
36     tot = 0
37     for k in range(round(-1.4 * x), round(1.4 * x)):
38         i, j = round(s * cos(theta) - k * sin(theta) + x / 2), round(s * sin(theta) + k * cos(theta) + x / 2)
39         if i >= 0 and i < x and j >= 0 and j < y:
40             tot += im[j, i]
41     return int(tot)
42
43 def sinogrammeV2(im, M=50):
```

Annexe - Code méthode par la transformée de Fourier discrète X

```
44     """ Realisation du sinogramme """
45     projections = []
46     for m in range(0, M):
47         lst = []
48         for k in range(-int(x / 2), int(x / 2) + 1):
49             lst.append(coupe_polaire(im, -m * pi / M, k))
50         projections.append(lst)
51     return np.vstack(projections) # on empile les
52     projections
53
54 # %% On propose une dernière méthode avec la fonction rotate
55
56 def radon(image, steps):
57     """ Meilleure méthode pour faire le sinogramme """
58     projections = []
59     dTheta = -180.0 / steps
```

Annexe - Code méthode par la transformée de Fourier discrète XI

```
60
61     for i in range(steps):
62         # on tourne l'image et on somme la verticale plutot
63         # que de sommer sur des droites qu'on fait tourner
64         projections.append(rotate(image, i * dTheta).sum(
65             axis=0)) # on somme sur l'axe verticale
66
67
68 # %% Algorithme de reconstruction sans filtrage :
69
70 def reverse(im):
71     xlen = im.shape[0]
72     ylen = im.shape[1]
73     resultat = np.zeros((ylen, ylen))
```

Annexe - Code méthode par la transformée de Fourier discrète XII

```
74     dTheta = 180.0 / xlen
75     for i in range(xlen):
76         temp = np.tile(im[i], (ylen, 1))
77         temp = rotate(temp, dTheta * i)
78         resultat += temp
79     return resultat
80
81
82 # %% Avec le filtrage :
83
84 def Lambda(N, M):
85     """ Filtre rampe """
86     t = np.zeros((N, M))
87     for k in range(N):
88         t[k, :] = abs(pi * TF.fftfreq(M, 2 / N))
89     return t
90
```

Annexe - Code méthode par la transformée de Fourier discrète XIII

```
91
92 def Q(im, N, M):
93     A = Lambda(N, M)    # Filtre
94     B = TF.fft(im, axis=1)  # Transformee de Fourier de l'
95     image (sur les lignes)
96     C = A * B  # Multiplication de B par le filtre (on
97     attenue les basse frequence et on augmente les haute
98     frequence)
99     return TF.ifft(C, axis=1)  # Transformee de Fourier
100    inverse
101
102
103 def reverse2(im):  # avec filtrage
104     xlen = im.shape[0]
105     ylen = im.shape[1]
106     im = Q(im, xlen, ylen).real  # On applique la fonction
107     inverse au sinogramme filtre
```

Annexe - Code méthode par la transformée de Fourier discrète XIV

```
03     resultat = np.zeros((ylen, ylen))
04     dTheta = 180.0 / xlen
05     for i in range(xlen):
06         temp = np.tile(im[i], (ylen, 1))
07         temp = rotate(temp, dTheta * i)
08         resultat += temp
09     return resultat
```

Plan

1 Introduction

- La Transformée de Radon
- La loi de Beer-Lambert

2 La reconstruction tomographique

- Utilisation des sinogrammes
- La résolution analytique
 - La rétroprojection simple
 - La rétroprojection filtrée

3 La résolution discrète

- Une première méthode : la transformée de Fourier discrète
- Une deuxième méthode : reconstruction algébrique

4 Les résultats obtenus avec les codes Python

5 Annexe

- La transformée de Fourier
- La transformée de Fourier discrète
- Code Python : méthode par la transformée de Fourier discrète
- Code Python : méthode ART

Annexe - Code méthode ART I

```
1 from math import cos, sin, sqrt, pi
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 mat = np.random.randint(10, size=(20, 20))
6 l = mat.shape[0] # nombre de ligne (coordonnee y)
7 L = mat.shape[1] # nombre de colonne (coordonnee x)
8 M = l * L
9 d = sqrt(l ** 2 + L ** 2)
10
11 # donne le point d'intersection de deux droites donnees en
12 # polaire
13 def distance(point1, point2):
14     return sqrt((point1[0] - point2[0]) ** 2 + (point1[1] -
15     point2[1]) ** 2)
16
17 # produit scalaire canonique
18 def ps(m1, m2):
```

Annexe - Code méthode ART II

```
18     return float((sum([i * j for (i, j) in zip(m1, m2)])))
19
20
21 # coordonées du pixel j
22 def pixel(j):
23     r = (j - 1) % L  # reste dans la division euclidienne de
24     # j-1 par L
25     q = (j - 1) // L  # quotient dans la division
26     # euclidienne de j-1 par L
27     return (q, r)
28
29
30
31 # conversion de l'image en une matrice colonne :
32 def colonne():
33     m = np.zeros((M, 1))
34     for j in range(0, M):
35         (q, r) = pixel(j + 1)
36         m[j, 0] = mat[q, r]
37
38     return m
```

Annexe - Code méthode ART III

```
35
36
37 # equation polaire de la droite
38 def rayon(u, t, theta):
39     return (u * sin(theta) + t * cos(theta), u * cos(theta)
40             - t * sin(theta))
41
42 # est ce que la droite de parametres (u,theta) passe par le
43 # pixel j
44
45 def isInPixel(u, theta, j):
46     (a, b) = pixel(j)
47     if theta != 0 and theta != pi / 2:
48         # Cas 1 (haut) :
49         t = (a - u * sin(theta)) / cos(theta)
50         y2 = rayon(u, t, theta)[1]
51         if b <= y2 <= b + 1:
52             return True
```

Annexe - Code méthode ART IV

```
52     # Cas 2 (bas) :
53     t = (a + 1 - u * sin(theta)) / cos(theta)
54     y2 = rayon(u, t, theta)[1]
55     if b <= y2 <= b + 1:
56         return True
57     # Cas 3 (gauche):
58     t = (u * cos(theta) - b) / sin(theta)
59     x2 = rayon(u, t, theta)[0]
60     if a <= x2 <= a + 1:
61         return True
62     # Cas 4 (droite):
63     t = (u * cos(theta) - b - 1) / sin(theta)
64     y2 = rayon(u, t, theta)[0]
65     if a <= x2 <= a + 1:
66         return True
67 elif theta == 0:
68     if a <= u <= a + 1:
69         return True
70 elif theta == pi / 2:
```

Annexe - Code méthode ART V

```
71         if b <= u <= b + 1:  
72             return True  
73     return False  
74  
75  
76 def intersection(u, theta, j):  
77     long = 0  
78     if not isInPixel(u, theta, j):  
79         return False  
80     (a, b) = pixel(j)  
81     if theta != pi / 2 and theta != 0:  
82         lst = []  
83         lst2 = []  
84         # intersection avec le bord gauche :  
85         t = (u * cos(theta) - b) / sin(theta)  
86         lst.append(rayon(u, t, theta))  
87  
88         # intersection avec le bord droit :  
89         t = (u * cos(theta) - b - 1) / sin(theta)
```

Annexe - Code méthode ART VI

```
90     lst.append((rayon(u, t, theta)))  
91  
92     # intersection avec le bord haut :  
93     t = (a - u * sin(theta)) / cos(theta)  
94     lst.append((rayon(u, t, theta)))  
95  
96     # intersection avec le bord bas :  
97     t = (a + 1 - u * sin(theta)) / cos(theta)  
98     lst.append((rayon(u, t, theta)))  
99     for point in lst[0:2]:  
00         if a <= point[0] <= a + 1:  
01             lst2.append(point)  
02     for point in lst[2:4]:  
03         if b <= point[1] <= b + 1:  
04             lst2.append(point)  
05     assert len(lst2) == 2  
06     long = distance(lst2[0], lst2[1])  
07 else:  
08     long = 1
```

Annexe - Code méthode ART VII

```
09     return long
10
11
12 # definition des rayons de projections
13 def projections(nb_dir, nb_droite):
14     dir = np.linspace(0, pi / 2, nb_dir)
15     droite = np.linspace(5, 15, nb_droite)
16     proj = []
17     for theta in dir:
18         for u in droite:
19             proj.append((u, theta))
20     return proj
21
22
23 def matR_v1(proj):
24     N = len(proj)
25     R = np.zeros((N, M))
26     for i in range(1, N + 1):
27         for j in range(1, M + 1):
```

Annexe - Code méthode ART VIII

```
28         if isInPixel(proj[i - 1][0], proj[i - 1][1], j):
29             R[i - 1, j - 1] = 1
30
31
32
33 # définition de la matrice R - version 2
34 def matR_v2(proj):
35     N = len(proj)
36     R = np.zeros((N, M))
37     for i in range(1, N + 1):
38         for j in range(1, M + 1):
39             R[i - 1, j - 1] = intersection(proj[i - 1][0],
proj[i - 1][1], j)
40
41
42
43 # construction de la matrice de projection
44 def matP(R):
45     m = colonne()
```

Annexe - Code méthode ART IX

```
46 N = R.shape[0]
47 P = np.zeros((N, 1))
48 for i in range(0, N):
49     for j in range(0, M):
50         P[i, 0] += R[i, j] * m[j, 0]
51 return P
52
53
54 # On peut alors tenter de résoudre le problème inverse : RF
55 = P (par n itération)
56 def opérateurs(R):
57     # extractions des lignes de la matrice R
58     N = R.shape[0]
59     lignes = []
60     transpo = []
61     q = []
62     for i in range(N):
63         lignes.append(R[i, :])
64         Ni = np.transpose(np.array([R[i, :]]))
```

Annexe - Code méthode ART X

```
64     transpo.append(Ni)
65     q.append(P[i, 0] * Ni / ps(Ni, Ni))
66
67 # definition des operateurs de projections orthognales
68 operator = []
69 id = np.eye(M)
70 for i in range(N):
71     Ni = transpo[i]
72     Ti = id - (1 / ps(Ni, Ni)) * np.dot(Ni, np.transpose(Ni))
73     operator.append(Ti)
74 return [q, operator]
75
76
77 def reconstruction(n, op, F_0=np.zeros((M, 1))):
78     nb = len(op[0])
79     T = op[1]
80     q = op[0]
81     F = F_0
```

Annexe - Code méthode ART XI

```
82     for k in range(1, n):
83         F = q[(k - 1) % nb] + np.dot(T[(k - 1) % nb], F - q
84                                     [(k - 1) % nb])
85         return F
86
87 proj = projections(500, 100)
88 R = matR_v1(proj)
89 P = matP(R)
90 op = operateurs(R)
91 FC = reconstruction(500, op)
92
93
94 def columnIntoMatrix(C):
95     A = np.zeros((1, L))
96     for i in range(1):
97         for j in range(L):
98             A[i, j] = C[i * L + j, 0]
99     return A
```

Annexe - Code méthode ART XII

```
00 F = columnIntoMatrix(FC)
01 plt.imshow(F)
02 plt.show()
03 plt.imshow(mat)
04 plt.show()
05
```