Reconstruction d'images avec l'algorithme ART

Rémi Germe

Juin 2022

1 Introduction

L'imagerie médicale 2D permet de détecter des anomalies, comme des tumeurs, en permettant de restituer des vues en coupe du corps des patients. Pour obtenir de telles images, on peut faire passer un scanner au patient pour récupérer des *données de projection*, permettant de reconstruire une vue de la zone scannée.

Le principe physique [1] est le suivant : on envoie des rayons x à travers le corps du patient, les rayons sont atténués en fonction des milieux qu'ils traversent (chair, organes, os...), et on mesure leur atténuation à la sortie du corps du patient. Concrètement, les données de projection sont la manière dont on trace les rayons dans le plan, ainsi que les atténuations mesurées. L'objectif est alors de reconstituer une image représentant l'atténuation de chaque point.

Il est possible d'employer des méthodes analytiques [2], cependant par la suite, nous nous intéresserons exclusivement à l'algorithme ART (*Algebraic Reconstruction Technique*). Enfin, nous ne prendrons pas en compte la présence de bruit, ce qui est certes peu réaliste, mais simplifie l'étude du problème.

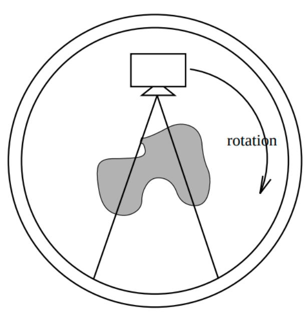


Figure 1: Principe physique : les rayons sont envoyés par un système tournant autour du patient. Image extraite de [1].

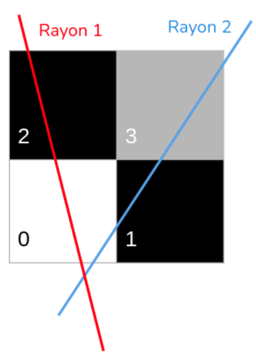
2 Discrétisation du problème

On représente une image de $p \in \mathbb{N}^*$ pixels par un vecteur $F = \begin{pmatrix} f_0 \\ ... \\ f_{p-1} \end{pmatrix} \in \mathbb{R}^p$, où f_j est le l'atténuation du j-ème pixel ($j \in \{0, ..., p-1\}$).

On trace $q \in \mathbb{N}^*$ rayons, numérotés de 0 à q - 1, auxquels on peut associer la *matrice de projection*, qui renseigne sur la manière dont été tracés les rayons : $A = (\delta_{i,j})_{(i,j) \in \{0,...,q-1\} \times \{0,...,p-1\}}$ où $\delta_{(i,j)}$ vaut 1 si le rayon i touche le pixel j, 0 sinon. On note $A_0, ..., A_{q-1}$ les lignes de A.

Enfin, le *vecteur de mesures*, qui contient les données mesurées par le récepteur, est $R = \begin{pmatrix} r_0 \\ ... \\ r_{q-1} \end{pmatrix} \in \mathbb{R}^q$, où r_i est l'atténuation totale du rayon $i \in \{0, ..., q-1\}$.

L'atténuation d'un rayon étant la somme des atténuations des pixels qu'il traverse, le problème s'écrit sous la forme $\mathbf{AF} = \mathbf{R}$. L'objectif est alors de déterminer F à partir de la donnée de A et R.



$$\mathrm{F} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0,5 \end{pmatrix} \quad \mathrm{R} = \begin{pmatrix} 1 \\ 1,5 \end{pmatrix} \quad \mathrm{A} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

Figure 2: Un exemple de discrétisation, où les atténuations sont comprises entre 0 (en noir) et 1 (en blanc).

3 L'algorithme ART

On veut résoudre le système linéaire $AF = R$ d'inconnue $F \in \mathbb{R}^p$. Malheureusement, en pratique ce système est largement sous-déterminé *ie* on a beaucoup plus d'inconnues scalaires (les f_j) que d'équations, ce qui correspond à $p > q$.

Intuitivement, l'algorithme ART consiste à reconstruire l'image itérativement, en ne prenant en compte que la contribution d'un certain rayon pour chaque itération.

Plus formellement, chaque ligne i du système définit un hyperplan affine π_i, dont $A_i^\top$ est un vecteur normal. On note H_i la direction de π_i et p_{π_i} (respectivement p_{H_i}) le projecteur orthogonal sur π_i (respectivement H_i).

L'idée intuitive de prendre en compte la contribution du rayon revient alors à projeter l'image partiellement reconstruite sur les π_i. On introduit alors $\sigma : \mathbb{N} \to \{0, ..., q-1\}$ tel que pour tout $n \in \mathbb{N}$, $\sigma(n)$ est le rayon choisi pour l'itération n (en numérotant à partir de 0).

En choisissant une image initiale $F_0 \in \mathbb{R}^p$, et en notant $\|.\|$ la norme euclidienne canonique, l'itération de l'algorithme s'écrit alors :

$$\begin{aligned}
\forall n \in \mathbb{N},\ F_{n+1} &= p_{\pi_{\sigma(n)}}(F_n) \\
&= p_{\pi_{\sigma(n)}}(0) + p_{H_{\sigma(n)}}(F_n) \\
&= p_{\pi_{\sigma(n)}}(0) + F_n - \frac{\langle F_n, A_{\sigma(n)}^\top \rangle}{\|A_{\sigma(n)}^\top\|^2} A_{\sigma(n)}^\top
\end{aligned}$$

Remarquons que pour tout i, $\pi_i = t_i + H_i$ avec $t_i = p_{\pi_i}(0)$. Or t_i est colinéaire à $A_i^\top$, il existe donc $\lambda_i \in \mathbb{R}$, $t_i = \lambda_i A_i^\top$. De plus $t_i \in \pi_i$, donc il vérifie $A_i t_i = r_i$ *ie* $\lambda_i A_i A_i^\top = \lambda_i \|A_i\|_2^2 = r_i$. A_i n'est pas nul (le rayon i passe forcément par au moins un pixel), alors $\lambda_i = \frac{r_i}{\|A_i\|^2}$. Ainsi, les t_i sont déterminés par les données de projection.

De plus, on appelle *schéma d'accès* l'ordre dans lequel on considère les rayons. Par la suite, on envisagera deux situations : les rayons seront pris successivement dans l'ordre dans lequel ils ont été tracés, on parlera alors du *schéma d'accès successif*, ou bien ils seront choisis aléatoirement, chacun avec une probabilité $P_i > 0$ indépendante de l'itération n, dans le cadre d'un *schéma d'accès aléatoire*.

Pour les schémas d'accès ci-dessus, on montre par la suite que l'algorithme converge, et ce vers une limite indépendante du schéma d'accès considéré. Il est cependant possible de montrer que le schéma d'accès influe sur la vitesse de convergence, et ce même pour des systèmes bruités [3].

4 Étude de la convergence de l'algorithme

4.1 Lemme

Lemme. Soit E un espace euclidien et $(u_n)_{n\in\mathbb{N}} \in E^{\mathbb{N}}$. On suppose que $(u_n)_{n\in\mathbb{N}}$ est bornée et qu'il existe $l \in E$ tel que toute suite extraite convergente de $(u_n)_{n\in\mathbb{N}}$ converge vers l. Alors $(u_n)_{n\in\mathbb{N}}$ converge vers l.

Preuve. Supposons par l'absurde que $(u_n)_{n\in\mathbb{N}}$ ne converge pas vers l. Alors il existe $\epsilon > 0$ tel que pour tout $N \in \mathbb{N}$, il existe $n \geq N$ tel que $|u_n - l| > \epsilon$. On dispose alors d'une infinité de rangs $n \in \mathbb{N}$ vérifiant $|u_n - l| > \epsilon$, on peut donc construire une extraction ϕ tel que pour tout $n \in \mathbb{N}$, $|u_{\phi(n)} - l| > \epsilon$.

Par hypothèse, $(u_\phi(n))_{n\in\mathbb{N}}$ est bornée, et possède donc une suite extraite convergente $(u_{\phi\circ\psi(n)})_{n\in\mathbb{N}}$ d'après le théorème de Bolzano-Weierstrass. De nouveau par hypothèse, $(u_{\phi\circ\psi(n)})_{n\in\mathbb{N}}$ converge vers l. Or les éléments de cette suite sont tous à une distance strictement plus grande que ϵ de l : contradiction. Ainsi, $(u_n)_{n\in\mathbb{N}}$ converge bien vers l.

4.2 Schéma d'accès successif

Convergence du schéma d'accès successif. On considère les rayons dans l'ordre dans lequel ils ont été tracés, *ie* pour tout $n \in \mathbb{N}$, $\sigma(n) = n \mod q$. Alors $(F_n)_{n\in\mathbb{N}}$ converge vers le projeté orthogonal L de F_0 sur $\bigcap_{i=0}^{q-1} \pi_i$.

Preuve. 1. Pour tout $n \in \mathbb{N}$, $\|\overrightarrow{F_nL}\|^2 = \|\overrightarrow{F_nF_{n+1}}\|^2 + \|\overrightarrow{F_{n+1}L}\|^2$ d'après le théorème de Pythagore. Donc $\|\overrightarrow{F_nL}\|^2 \geq \|\overrightarrow{F_{n+1}L}\|^2$, la suite $(\|\overrightarrow{F_nL}\|^2)_{n\in\mathbb{N}}$ est décroissante et bornée. D'après le théorème de la limite monotone, $(\|\overrightarrow{F_nL}\|^2)_{n\in\mathbb{N}}$ converge vers un certain réel $\delta \geq 0$. Alors en passant à la limite, on obtient : $\lim_{n\to\infty} \|\overrightarrow{F_nF_{n+1}}\|^2 = \delta - \delta = 0$.

2. D'après ce qui précède, $(F_n)_{n\in\mathbb{N}}$ est bornée. Pour montrer que $(F_n)_{n\in\mathbb{N}}$ converge vers L, il nous suffit alors de montrer que toutes ses suites extraites convergentes convergent vers L. Soit $(F_{\phi(n)})_{n\in\mathbb{N}}$ une suite extraite convergente, de limite $L' \in \mathbb{R}^p$.

3. L'image de ϕ est une partie infinie de $\mathbb{N}$, donc son intersection avec $q\mathbb{Z} + i$ est infinie pour un certain $i \in \{0, ..., q-1\}$. Nous pouvons extraire de $(F_{\phi(n)})_{n\in\mathbb{N}}$ une suite $(F_{\psi(n)})_{n\in\mathbb{N}}$ telle que pour tout $n \in \mathbb{N}$: $\psi(n) \mod q = i$. Il en découle par continuité de p_{π_i} que : $F_{\psi(n)+1} = p_{\pi_i}(F_{\psi(n)}) \underset{n\to\infty}{\longrightarrow} p_{\pi_i}(L')$. Or : $\|\overrightarrow{L'p_{\pi_i}(L')}\| = \lim_{n\to\infty} \|\overrightarrow{F_{\psi(n)}F_{\psi(n)+1}}\| = 0$, donc : $p_{\pi_i}(L') = L'$ et $L' \in \pi_i$.

4. La suite $(F_{\psi(n)+1})_{n\in\mathbb{N}}$ converge maintenant vers L' et nous allons pouvoir répéter le raisonnement qui précède. On note abusivement $p_{\pi_{i+1}}$ le projecteur $p_{\pi_{(i+1) \mod q}}$.Par définition de ψ et continuité de $p_{\pi_{i+1}}$: $F_{\psi(n)+2} = p_{\pi_{i+1}}(F_{\psi(n)+1}) \underset{n\to\infty}{\longrightarrow} p_{\pi_{i+1}}(L')$. Or : $\|\overrightarrow{L'p_{\pi_{i+1}}(L')}\| = \lim_{n\to\infty} \|\overrightarrow{F_{\psi(n)+1}F_{\psi(n)+2}}\| = 0$, donc : $p_{\pi_{i+1}}(L') = L'$ et $L' \in \pi_{i+1}$. De proche en proche : $L' \in \pi_0 \cap ... \cap \pi_{q-1}$, ou encore : $\overrightarrow{LL'} \in H_0 \cap ... \cap H_{q-1}$.

5. Puis, pour tout $n \in \mathbb{N}$: $F_n = F_0 + \overrightarrow{F_0F_1} + \overrightarrow{F_1F_2} + ... + \overrightarrow{F_{n-1}F_n} \in F_0 + H_0^\perp +$

$...+H_{q-1}^{\perp} = F_0+(H_0\cap...\cap H_{q-1})^{\perp}$, or le sous-espace affine $F_0+(H_0\cap...\cap H_{q-1})^{\perp}$ est fermé donc $L' = \lim_{n\to\infty} F_{\phi(n)} \in F_0 + (H_0 \cap ... \cap H_{q-1})^{\perp}$. A fortiori : $\overrightarrow{LL'} \in \overrightarrow{LF_0} + (H_0 \cap ... \cap H_{q-1})^{\perp} = (H_0 \cap ... \cap H_{q-1})^{\perp}$ car $\overrightarrow{LF_0} \in (H_0 \cap ... \cap H_{q-1})^{\perp}$. Finalement, grâce au point précédent : $\overrightarrow{LL'} = 0$, *ie* $L = L'$.

4.3 Schéma d'accès aléatoire

On introduit une suite de variables aléatoires $(X_n)_{n\in\mathbb{N}}$ indépendantes et identiquement distribuées, définies sur un même univers Ω et à valeurs dans $\{0,...,q-1\}$, modélisant le choix aléatoire d'un rayon à l'itération n, c'est-à-dire telles que pour tout $i \in \{0,...,q-1\}$, $\mathbb{P}(X_n = i) = P_i$, où P_i est la probabilité de choisir le rayon i.

Ici, on considère la fonction $\sigma_{alea} : \omega \mapsto \sigma_w = (n \mapsto X_n(w))$ définie sur Ω. Par la suite, on ne notera pas les ω, et on écrira abusivement : pour tout $n \in \mathbb{N}, \sigma(n) = X_n$.

Convergence des schémas d'accès aléatoires. Sous les hypothèses ci-dessus, $(F_n)_{n\in\mathbb{N}}$ converge presque sûrement vers L.

Preuve. On peut reprendre les points 1, 2 et 5 de la preuve précédente. Il nous suffit alors de montrer que $L' \in \pi_0 \cap ... \cap \pi_{q-1}$, en reprenant les notations précédemment utilisées.

L'objectif est de trouver, pour tout $i \in \{0,...,q-1\}$, une infinité de rangs n tels que $X_{\phi(n)} = i$. On pourra alors extraire une suite $(F_{\psi(n)})_{n\in\mathbb{N}}$ telle que pour tout $n \in \mathbb{N}, X_{\psi(n)} = i$. Alors l'argument utilisé en 3 pour conclure que $L' \in \pi_i$ s'appliquera, et on obtiendra : $L' \in \pi_i$.

Pour tout $i \in \{0,...,q-1\}$, on note $A_i = \bigcap_{N\in\mathbb{N}} \bigcup_{n\geq N} [X_{\phi(n)} = i]$. Montrons que A_i est presque sûr. Une intersection dénombrable d'événements presque sûrs étant presque sûre, il suffit de montrer que pour tout $N \in \mathbb{N}, B_{i,N} = \bigcup_{n\geq N} [X_{\phi(n)} = i]$ est presque sûr. Montrons alors que $\overline{B_{i,N}}$ est négligeable. Pour tout $N_1 \geq N$:

$$\begin{aligned}
\mathbb{P}(\overline{B_{i,N}}) &\leq \mathbb{P}(\bigcap_{n=N}^{N_1} \overline{[X_{\phi(n)} = i]}) \\
&\leq \prod_{n=N}^{N_1} \mathbb{P}(\overline{X_{\phi(n)} = i}) \text{ par indépendance mutuelle des } X_{\phi(n)} \\
&\leq \prod_{n=N}^{N_1} (1 - P_i) = (1 - P_i)^{N_1-N+1} \underset{N_1\to\infty}{\longrightarrow} 0 \text{ car } P_i > 0
\end{aligned}$$

Ainsi $B_{i,N}$ est presque sûr, et ce pour tout $N \in \mathbb{N}$. Finalement A_i est presque sûr, donc $L' \in \pi_i$ presque sûrement. De nouveau, une intersection (finie ici) d'événements presque sûrs est presque sûre donc $L' \in \bigcap_{i=0}^{q-1} \pi_i$ presque sûrement.

5 Simulation informatique

J'ai implémenté l'algorithme ART pour vérifier les prévisions théoriques établies ci-dessus, ainsi que pour étudier l'influence du schéma d'accès choisi sur la vitesse de convergence.

Pour cela, on part d'une *image cible* F, on trace des rayons et on détermine A, puis on calcule $R = AF$. L'ART prend en argument les données de projection, *ie* A et R.

Les atténuations seront représentées par des flottants compris entre 0 (pixel noir) et 1 (pixel blanc).

Pour évaluer la qualité d'une image I reconstruite à partir des données d'une image cible F, on utilise deux outils quantitatifs :

- l'*écart-moyen* : $\delta_1(F, I) = \frac{\|F-I\|_1}{p}$

- l'*écart en norme 2* : $\delta_2(F, I) = \|F - I\|_2$

Enfin, on définit le *nombre de cycles* comme étant le rapport du nombre d'itérations effectuées sur le nombre de rayons utilisés.

Figure 3: On utilise deux images cibles pour tracer les graphiques : à gauche, le *Shepp Logan phantom* (SLP), image standard de test en tomographie, à droite, un lapin.

On considère différents schémas d'accès : le schéma successif, un schéma aléatoire uniforme ($P_i = \frac{1}{q}$), ainsi qu'un schéma aléatoire "amélioré" ($P_i = \frac{\|A_i\|^2}{\|A\|_2^2}$), dont il est prouvé qu'il converge exponentiellement vite dans le cas des systèmes surdéterminés ($q > p$, ce qui n'est pas le cas ici) [3].

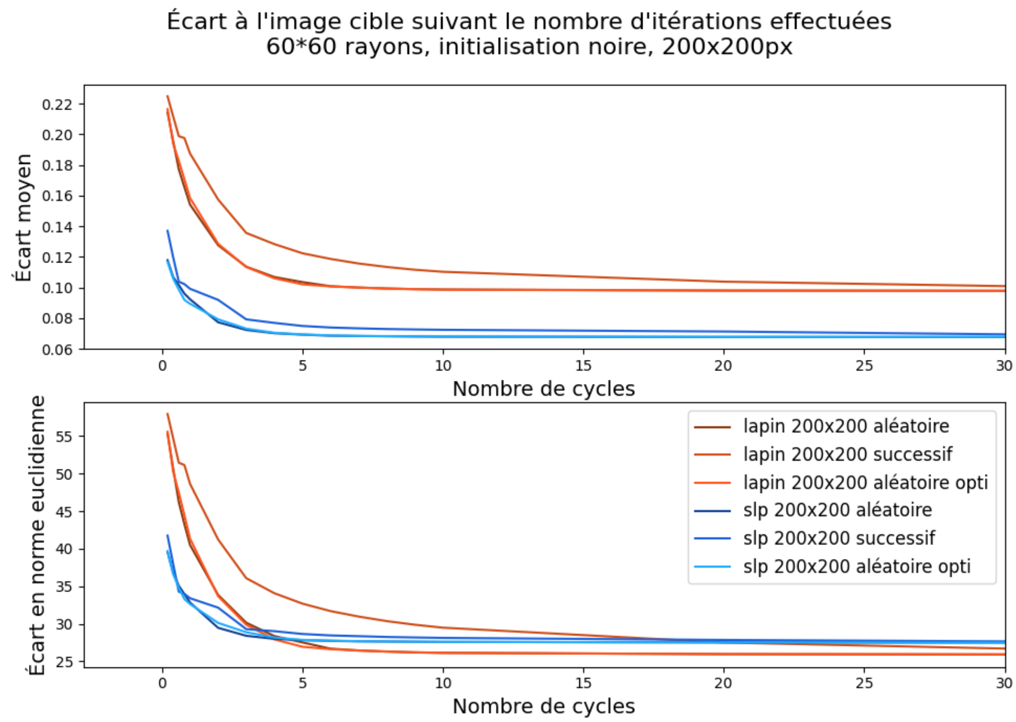


Figure 4: Ecart à l'image cible pour chacune des deux images de test (taille 200x200 px) en fonction du nombre de cycles effectués, suivant différents schémas d'accès

On a bien convergence pour chacun des schémas d'accès - avec une convergence légèrement plus rapide pour les schémas aléatoires. On vérifie que les images reconstruites asymptotiquement par les différents schémas d'accès sont les mêmes, ce qui est bien le cas.

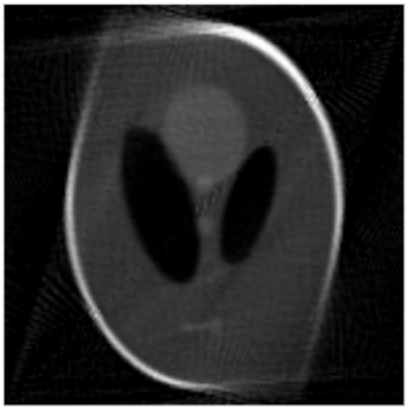
 

Figure 5: Reconstitution d'images 150x150px, à partir d'une initialisation noire, 120x120 rayons, +50 cycles (asymptotique)

A Annexe

A.1 Code de la simulation informatique

```
from numpy import tan, cos, sin, pi, zeros, linspace, vdot, dot
from numpy.linalg import norm
from random import choices, randint
from dataclasses import dataclass
from PIL import Image


#            #
# Resolution #
#            #

def prochain_rayon(j, N_R, aleatoire, P, les_rayons):
    """
    aleatoire : 0 pour schema successif
                1 pour aleatoire equiprobable
                2 pour aleatoire optimise :
                    on prend le rayon j avec une probabilite
                    de norme2(A[j])**2 / norme_euc(A) ** 2
                    ces probabilites sont contenues dans la liste P
    """
    # j est le dernier rayon utilise
    if aleatoire == 0:
        return (j + 1) % N_R
    elif aleatoire == 1:
        return randint(0, N_R - 1)
    return choices(les_rayons, weights = P, k = 1)[0]


def ART(f0, A, R, N_ITER, aleatoire, cst):
```

```python
    N_R = cst.N_THETA * cst.N_RHO # nombre de rayons
    N_P = cst.L * cst.H # nombre de pixels
    # Les vecteurs normaux aux hyperplans sont les (lignes) A[j]
    # Il est pratique de les rendre unitaires
    # On calcule la norme 2 de chaque ligne
    N = zeros((N_R, N_P))
    normes = norm(A, ord = 2, axis = 1)
    for j in range(N_R):
        N[j] = A[j] / normes[j]

    # Pour le choix aleatoire optimise des rayons
    norme_A = norm(A)
    P = [(normes[j] / norme_A) ** 2 for j in range(N_R)]
    les_rayons = list(range(0, N_R))

    # On calcule aussi les projections orthogonales de 0 sur les hyperplans
    T = zeros((N_R, N_P))
    for j in range(N_R):
        T[j] = R[j] / normes[j] * N[j]

    # Initialisation
    f = f0
    j = 0
    for _ in range(N_ITER):
        f = f - vdot(f, N[j]) * N[j] + T[j]
        j = prochain_rayon(j, N_R, aleatoire, P, les_rayons)
    tronquer(f)
    return f

#                      #
# Exemple d'utilisation #
#                      #

def main():
    I, L, H = image_depuis_fichier("exemples/lapin/lapin_100.png")
    cst = Donnees(L, H, 80, 80)
    A = tracer_rayons(cst)
    print("Rayons traces")

    R = dot(A, I) # les mesures ne sont pas bruitees
    f0 = zeros(L * H)
    f = ART(f0, A, R, 6 * 80 * 80 + 1, 1, cst) # 6 cycles
    print(ecart_moyen(f, I, L * H))
    print(ecart_norme_euc(f, I))
    enregistrer_image(f, L, H, "lapin_200_80_80_6_cycles_alea.png")

main()


#              #
# Utilitaires #
#              #

@dataclass
class Donnees:
    L: int # la largeur de l'image
    H: int # la hauteur de l'image
```

```
    N_THETA: int # le nombre d'angles utilises pour tracer les rayons
    N_RHO: int # le nombre de rayons traces par angle


def indice_pixel(x: int, y: int, H):
    # On numerote de bas en haut, et de gauche a droite
    # On prend les coordonnees PIL
    return x * H + y

def coords_pixel(k, H):
    return (k // H, k % H)


def tronquer(f):
    # Pour avoir des valeurs dans [0, 1]
    for i in range(len(f)):
        f[i] = max(0, min(1, f[i]))


def ecart_moyen(f, g, N_P):
    return norm(f - g, ord = 1) / N_P

def ecart_norme_euc(f, g):
    return norm(f - g, ord = 2)


def image_depuis_fichier(fichier):
    # On renvoie le vecteur ainsi que sa taille
    im = Image.open(fichier)
    # On convertit l'image en mode noir et blanc
    if im.mode != "L":
        im = im.convert("L")
    L, H = im.size
    N_R = L * H
    I = zeros(N_R, dtype = float)
    for k in range(N_R):
        x, y = coords_pixel(k, H)
        I[k] = im.getpixel((x, y)) / 255
    return (I, L, H)

def enregistrer_image(f, L, H, fichier):
    im = Image.new("L", (L, H), color = 0)
    for k in range(L * H):
        x, y = coords_pixel(k, H)
        im.putpixel((x, y), int(f[k] * 255))
    im.save(fichier)


#                   #
# Tracage des rayons #
#                   #


def est_valide(x, y, L, H):
    return 0 <= x < L and 0 <= y < H

"""
```

```
On parcourt le plan de gauche a droite pour determiner les pixels touches par un rayon
En ne testant que les trois voisins potentiellement touches par le rayon a chaque fois
(trois voisins car on sait si la droite affine qu'est le rayon a un coefficient directeur
positif (haut) ou negatif (bas))
"""

# On progresse vers la droite
# On separe le plan en deux
Haut = 0
Bas = 1

# En fonction de la direction, on doit
# verifier les 3 cases adjacentes
Directions = [
    [(0, -1), (1, -1), (1, 0)],
    [(1, 0), (1, 1), (0, 1)],
]

def rayon_touche_pixel(theta, rho, x, y):
    """
    Attention : ne s'occupe pas de savoir si le pixel est valide
    (x, y) : coordonnees du pixel
    f_k : f(x) avec f : x -> x * tan(theta) + rho / cos(theta)
    l'equation de la droite que parcourt le rayon
    """
    f_k = x * tan(theta) + rho / cos(theta)
    return (f_k < y and f_k + tan(theta) > y) \
            or (f_k > y + 1 and f_k + tan(theta) < y + 1) \
            or (y < f_k < y + 1)


def trouver_premier_pixel(theta, rho, L, H):
    # Renvoie le premier pixel touche ainsi que la direction

    # On determine la direction
    d = Bas if theta < pi / 2 else Haut

    # Fonction auxiliaire pour eviter de refaire le calcul de tan(theta)
    def _rayon_touche_pixel(y, f_k):
        return (f_k < y and f_k + tan_theta > y) \
                or (f_k > y + 1 and f_k + tan_theta < y + 1) \
                or (y < f_k < y + 1)

    f_k = rho / cos(theta)
    tan_theta = tan(theta)
    for x in range(0, L):
        if x == 0 or x == L - 1:
            # On essaie sur tout le cote de l'image
            for y in range(0, H):
                if _rayon_touche_pixel(y, f_k):
                    return ((x, y), d)
        else:
            # On essaie uniquement aux bords de l'image
            if _rayon_touche_pixel(0, f_k):
                return ((x, 0), d)
            if _rayon_touche_pixel(L - 1, f_k):
                return ((x, L - 1), d)
```

```
        f_k += tan_theta
    raise Exception("Un rayon trace ne passe par aucun pixel, n'est pas cense se produire")


def tracer_rayons(cst):
    # Renvoie la matrice de projection
    N_R = cst.N_THETA * cst.N_RHO
    N_P = cst.L * cst.H
    A = zeros((N_R, N_P), dtype = int)
    e = 0.05 # pour ne pas etre trop sur les bords

    for (i, theta) in enumerate(linspace(e, pi - e, cst.N_THETA)):
        rho_min = - cst.L * sin(theta)
        rho_max = cst.H * cos(theta)
        tan_theta = tan(theta) # pour eviter de le recalculer plein de fois
        for (j, rho) in enumerate(linspace(rho_min + e, rho_max - e, cst.N_RHO)):
            # Fonction auxiliaire pour eviter de faire trop de calculs
            def _rayon_touche_pixel(y, f_k):
                return (f_k < y and f_k + tan_theta > y) \
                    or (f_k > y + 1 and f_k + tan_theta < y + 1) \
                    or (y < f_k < y + 1)
            # Initialisation
            pas_vus = [True] * N_P # pour ne pas boucler sur des pixels deja visites
            ((x, y), d) = trouver_premier_pixel(theta, rho, cst.L, cst.H)
            directions = Directions[d]
            f_k = x * tan_theta + rho / cos(theta)
            ind = indice_pixel(x, y, cst.H)
            pas_vus[ind] = False
            A[cst.N_RHO * i + j][ind] = 1
            def prochain_pixel(x, y, f_k):
                # On parcourt de proche en proche les pixels
                # On regarde les trois suivants dans le demi-plan qui convient
                for (dx, dy) in directions:
                    x_ = x + dx
                    y_ = y + dy
                    ind = indice_pixel(x_, y_, cst.H)
                    if est_valide(x_, y_, cst.L, cst.H) and pas_vus[ind] and _rayon_touche_pix
                        A[cst.N_RHO * i + j][ind] = 1
                        pas_vus[ind] = False
                        f_k_p = f_k if dx == 0 else f_k + tan_theta
                        prochain_pixel(x_, y_, f_k_p)
            prochain_pixel(x, y, f_k)
    return A
```

References

[1] Isabelle Bloch. *Reconstruction d'images de tomographie*. URL: `https://perso.telecom-paristech.fr/bloch/ATIM/tomo.pdf`. Cours du Laboratoire Traitement et Communication de l'Information (LTCI), Télécom Paris.

[2] Ali MOHAMMAD-DJAFARI. *Transformée de Radon et reconstruction d'image*. 2002. URL: `http://djafari.free.fr/pdf/tr.pdf`. Cours du Laboratoire des Signaux et Systèmes, Centrale Supélec.

[3] Deanna Needell. "Randomized Kaczmarz solver for noisy linear systems". In: *Bit Numerical Mathematics* 50 (2010), pp. 395–403. DOI: `https://doi.org/10.1007/s10543-010-0265-5`.