UMONS
Université de Mons

Faculté Polytechnique

Projet d'Optimisation Linéaire

Suppression de l'arrière-plan dans une vidéo

BOREUX Tom
LEMPIRE Martin

POLYTECH MONS

Professeur : GILLIS Nicolas

5 Décembre 2025

pôle hainuyer

Table des matières

Chapitre 1

Introduction

L'analyse d'une séquence vidéo peut être abordée en considérant la matrice X dont chaque colonne représente une image et chaque ligne l'évolution temporelle d'un pixel. Dans de nombreux contextes, l'arrière-plan est supposé fixe tandis que l'avant-plan introduit des variations locales dues aux objets en mouvement. L'objectif est alors d'estimer une image de fond cohérente malgré ces perturbations.

Pour cela, nous modélisons le fond comme une image constante b qui doit se rapprocher au mieux des intensités observées dans X. L'utilisation de la norme 1 permet de limiter l'influence des valeurs aberrantes, fréquentes lorsque des objets se déplacent dans la scène. Cette approche conduit naturellement à un problème d'optimisation linéaire permettant d'obtenir une estimation robuste du fond, ensuite reconstruite sous la forme d'une image unique.

Ce rapport est structuré en plusieurs étapes :

1. Estimation de l'arrière plan
2. Estimation de l'illumination.
3. Reconstruction de l'avant-plan et résultat final.
4. Raffinement itératif de la solution.

Chapitre 2

Estimation de l'arrière-plan

2.1 Modélisation du problème

2.1.1 Un problème d'optimisation linéaire pour un pixel fixe

Pour chaque pixel i, la colonne $X(i,:)$ représente l'évolution de son intensité au cours des t images. Notre objectif est d'estimer une valeur constante b_i qui décrit au mieux l'arrière-plan pour ce pixel. Afin de mesurer l'écart entre b_i et les observations, nous utilisons la norme 1, ce qui conduit au problème suivant :

$$\min_{b_i \in \mathbb{R}} \sum_{j=1}^{t} |X(i,j) - b_i|.$$

Pour transformer ce problème en un programme linéaire, nous introduisons pour chaque j une variable $y_j \geq 0$ satisfaisant $y_j \geq |X(i,j) - b_i|$. Le problème devient alors :

$$\min_{b_i, y} \sum_{j=1}^{t} y_j$$

avec les contraintes :

$$y_j \geq X(i,j) - b_i, \qquad y_j \geq -(X(i,j) - b_i), \qquad y_j \geq 0.$$

2.1.2 Forme standard

Pour exprimer ce programme sous forme standard, toutes les contraintes doivent être des égalités et toutes les variables doivent être non négatives. Comme b_i est une variable libre, on le décompose en

$$b_i = b_i^+ - b_i^-, \qquad b_i^+ \geq 0, \quad b_i^- \geq 0.$$

Fonction objectif

$$\min_{b_i^+, b_i^-, y, s_1, s_2} \sum_{j=1}^{t} y_j.$$

Contraintes sous forme d'égalités

Les deux inégalités linéarisant la valeur absolue deviennent :

$$y_j + b_i^+ - b_i^- - s_{1j} = X(i,j),$$

$$y_j - b_i^+ + b_i^- - s_{2j} = -X(i,j),$$

où s_{1j} et s_{2j} sont des variables d'écart satisfaisant $s_{1j} \geq 0$, $s_{2j} \geq 0$.

Variables non négatives

Toutes les variables sont maintenant contraintes à être positives :

$$y_j \geq 0, \quad s_{1j} \geq 0, \quad s_{2j} \geq 0, \quad b_i^+ \geq 0, \quad b_i^- \geq 0.$$

2.2 Implémentation et résolution du problème à l'aide de Python

Dans cette partie, nous cherchons à reconstruire, pour chaque pixel i, une valeur optimale b_i à partir des colonnes de la matrice X. L'idée est de déterminer une valeur unique b_i qui soit cohérente avec toutes les intensités observées sur l'ensemble des images, en minimisant l'écart entre cette valeur et les données observées.

2.2.1 Préparation des données

On commence par charger la matrice X à partir du fichier `pedsX_il.mat` en utilisant la bibliothèque `scipy.io`. Cette matrice X a une taille de $p \times t$, où p est le nombre de pixels (lignes) et t est le nombre d'images (colonnes). on obtient ainsi 100 images de 152x232 pixels, soit un total de $p = 152 \times 232 = 35264$ pixels

2.2.2 Méthodologie

Pour chaque ligne de la matrice X, nous avons formulé un problème d'optimisation linéaire visant à minimiser la somme des écarts absolus entre la valeur reconstruite et les valeurs réellement observées. Concrètement, pour chaque pixel, nous cherchons une valeur unique b_i qui approxime au mieux l'ensemble des intensités mesurées dans les différentes colonnes de X, en introduisant des variables d'erreur positives permettant d'encadrer ces écarts.

Cette approche revient à ajuster individuellement chaque pixel en imposant un ensemble de contraintes linéaires garantissant que la différence entre la reconstruction et les données observées reste contrôlée, tout en minimisant la somme totale des erreurs. La résolution de ce problème est effectuée à l'aide de la fonction `linprog` de `scipy`, qui permet d'obtenir, pour chaque pixel, la valeur optimale contribuant à la reconstruction finale de l'image.

2.2.3 Résultats

La méthode d'optimisation appliquée pixel par pixel permet d'obtenir une estimation robuste de l'intensité moyenne de fond à partir des différentes images. L'algorithme fournit ainsi un vecteur b qui, une fois remis en forme, constitue une approximation du *background* de la scène.

Pour illustrer le résultat obtenu, nous présentons ci-dessous :
— la **première image de la vidéo**, telle qu'elle apparaît dans la matrice X ;
— l'**image de fond reconstruite** à partir du vecteur optimisé b.

Première image de la vidéo Image de fond reconstruite

FIGURE 2.1 – Comparaison entre l'image d'entrée et le fond estimé.

Ces figures permettent de visualiser clairement la différence entre une image contenant les objets (ici, les piétons) et la reconstruction du fond statique obtenue par notre méthode.

Pour plus de détails sur l'implémentation, veuillez consulter l'**Annexe A**, qui contient le code complet.

2.3 Sommet du polyèdre

Le programme linéaire associé à l'estimation du fond impose, pour chaque image j, les contraintes

$$e_j \geq |x_{ij} - b|.$$

L'ensemble des solutions admissibles forme un polyèdre convexe dans l'espace $(b, e_1, \ldots, e_t)$. Un sommet correspond à un point où suffisamment de contraintes deviennent actives. Dans ce problème, les contraintes actives sont de la forme

$$e_j = b - x_{ij}, \qquad e_j = x_{ij} - b, \qquad \text{ou} \qquad e_j = 0.$$

Un sommet est obtenu lorsque, pour au moins une image j, on impose

$$e_j = 0 \quad \Longrightarrow \quad b = x_{ij}.$$

Les autres erreurs s'ajustent alors automatiquement selon

$$e_k = |x_{ik} - b|, \qquad k \neq j.$$

Ainsi, les sommets du polyèdre sont exactement les points où b prend l'une des valeurs observées x_{ij}. L'optimum du programme linéaire se situe toujours sur l'un de ces sommets, ce qui conduit à choisir b égal à la médiane des observations.

2.4 La solution optimale de $\min_{b\in\mathbb{R}^p} \sum_{j=1}^{t} \|X(:,j)-b\|_1$ est la médiane : Explication

La solution optimale de $\min_{b\in\mathbb{R}^p} \sum_{j=1}^{t} \|X(:,j)-b\|_1$ est la médiane. En effet, la fonction objectif

$$f(b) = \sum_{j=1}^{t} \|X(:,j)-b\|_1$$

mesure la somme des distances absolues entre le vecteur du fond b et les observations $X(:,j)$. Pour chaque coordonnée (chaque pixel), ce problème se réécrit

$$\min_{b_i\in\mathbb{R}} \sum_{j=1}^{t} |x_{ij}-b_i|.$$

Une telle somme de valeurs absolues est une fonction convexe, en morceaux linéaire, dont la dérivée change de signe exactement lorsque le point b_i sépare les données en deux ensembles de même taille. Ce point est précisément la *médiane* des valeurs $\{x_{i1},\dots,x_{it}\}$.

Autrement dit, la médiane est le seul point pour lequel le nombre d'observations situées au-dessus et en dessous de b_i est identique, ce qui annule la dérivée généralisée de la fonction. Cela caractérise le minimum de la norme L^1.

Ainsi, pour chaque pixel i, la solution optimale est

$$b_i^\star = \mathrm{med}(x_{i1},\dots,x_{it}),$$

et l'assemblage de ces composantes fournit le vecteur optimal $b^\star$. La médiane apparaît donc naturellement comme la solution qui minimise la somme des écarts absolus.

Chapitre 3

Estimation de l'illumination

3.1 Modélisation du problème

3.1.1 Un problème d'optimisation linéaire

Après avoir estimé le vecteur b, nous cherchons à déterminer, pour chaque image j, un coefficient d'illumination a_j permettant d'ajuster la luminosité globale. En effet, bien que le fond soit fixe, les conditions d'éclairage peuvent varier au cours de la séquence (soleil, ombres, nuages,...).

Pour une image donnée, on compare chaque intensité $X(i,j)$ à la reconstruction $a_j b_i$. Les écarts sont mesurés au moyen de la norme 1, ce qui conduit à considérer le problème suivant :

$$\min_{a_j \in \mathbb{R}} \sum_{i=1}^{p} |X(i,j) - a_j b_i|.$$

Comme dans la section consacrée à l'estimation de b, on introduit pour chaque pixel une variable d'erreur $e_i \geq 0$ satisfaisant $e_i \geq |X(i,j) - a_j b_i|$. Le problème devient alors :

$$\sum_{i=1}^{p} e_i$$

avec les contraintes :

$$e_i \geq X(i,j) - a_j b_i, \qquad e_i \geq -(X(i,j) - a_j b_i). \qquad e_i \geq 0.$$

3.1.2 Forme standard

Pour exprimer ce problème sous forme standard, toutes les contraintes doivent être écrites sous forme d'égalités et toutes les variables doivent être non négatives. Comme a_j est une variable libre, on le décompose en

$$a_j = a_j^+ - a_j^-, \qquad a_j^+ \geq 0, \quad a_j^- \geq 0.$$

Fonction objectif

$$\min_{a_j^+, a_j^-, e, s_1, s_2} \sum_{i=1}^{p} e_i.$$

Contraintes sous forme d'égalités

Les deux inégalités linéarisant la valeur absolue deviennent :

$$e_i + a_j^+ b_i - a_j^- b_i - s_{1i} = X(i,j),$$
$$e_i - a_j^+ b_i + a_j^- b_i - s_{2i} = -X(i,j).$$

où s_{1i} et s_{2i} sont des variables d'écart satisfaisant $s_{1i} \geq 0$, $s_{2i} \geq 0$.

Variables non négatives

Toutes les variables sont maintenant contraintes à être positives :

$$e_i \geq 0, \quad s_{1i} \geq 0, \quad s_{2i} \geq 0, \quad a_j^+ \geq 0, \quad a_j^- \geq 0.$$

3.2 Implémentation et résolution du problème à l'aide de Python

Dans cette partie, nous cherchons à estimer, pour chaque image j, un coefficient d'illumination a_j qui ajuste au mieux l'intensité globale observée dans $X(:,j)$ à partir du vecteur de fond b obtenu précédemment. Comme pour l'estimation de b, le problème est traité colonne par colonne, ce qui permet de résoudre t problèmes indépendants à l'aide de programmes linéaires de petite taille.

3.2.1 Préparation des données

La matrice X étant déjà chargée dans la partie précédente, nous disposons d'une matrice de taille $p \times t$, où p est le nombre de pixels et t le nombre d'images. Le vecteur b obtenu dans la section précédente est utilisé comme référence pour ajuster chacune des colonnes $X(:,j)$.

3.2.2 Méthodologie

Pour chaque colonne de la matrice X, nous avons formulé un problème d'optimisation linéaire visant à ajuster un coefficient d'illumination unique a_j permettant de reproduire au mieux l'ensemble des intensités observées dans l'image correspondante. Concrètement, pour chaque image, nous cherchons une valeur scalaire a_j qui corrige globalement la luminosité en comparant les valeurs mesurées $X(i,j)$ aux valeurs reconstruites $a_j b_i$. Des variables d'erreur positives sont introduites pour encadrer ces écarts et garantir une modélisation linéaire de la norme 1.

Cette approche revient à ajuster individuellement chaque image en imposant un ensemble de contraintes linéaires assurant que la différence entre la reconstruction $a_j b$ et les données observées reste contrôlée pour tous les pixels. La minimisation de la somme totale des erreurs permet ensuite d'obtenir, pour chaque image, la valeur optimale du coefficient a_j. La résolution de ce problème est effectuée à l'aide de la fonction `linprog` du module `scipy`, qui fournit directement les coefficients d'illumination nécessaires à la reconstruction finale de la séquence.

3.2.3 Résultats

La méthode employée permet d'obtenir un vecteur $a \in \mathbb{R}^t$ décrivant l'évolution de la luminosité au cours de la séquence vidéo. Ces coefficients reflètent les variations observées dans les images originales et permettent, dans le chapitre suivant, de reconstruire l'avant-plan via

$$F = X - ba^\top.$$

3.3 Sommet du polyèdre

Le programme linéaire associé à l'estimation du coefficient d'illumination impose, pour chaque pixel i, les contraintes

$$e_i \geq |X(i,j) - a_j b_i|.$$

L'ensemble des solutions admissibles forme un polyèdre convexe dans l'espace $(a_j, e_1, \ldots, e_p)$. Comme dans le cas de l'estimation de b, un sommet correspond à un point où suffisamment de contraintes deviennent actives. Ici, les contraintes actives sont de la forme

$$e_i = X(i,j) - a_j b_i, \qquad e_i = a_j b_i - X(i,j), \qquad \text{ou} \qquad e_i = 0.$$

En particulier, une contrainte devient active lorsque

$$e_i = 0 \quad \Longrightarrow \quad X(i,j) = a_j b_i.$$

Dans ce cas, on en déduit directement

$$a_j = \frac{X(i,j)}{b_i}.$$

Les autres erreurs se déterminent alors automatiquement suivant

$$e_k = |X(k,j) - a_j b_k|, \qquad k \neq i.$$

Ainsi, les sommets du polyèdre correspondent exactement aux valeurs obtenues en prenant

$$a_j = \frac{X(i,j)}{b_i}$$

pour l'un des pixels i tels que $b_i \neq 0$. Comme dans le cas de l'estimation du vecteur b, l'optimum du programme linéaire se situe toujours sur l'un de ces sommets.

Chapitre 4

Résultat final

Une fois les coefficients d'illumination a et le vecteur d'arrière-plan b calculés, il est possible de reconstruire l'avant-plan en évaluant les écarts résiduels par rapport au modèle de rang 1. Pour chaque image j, l'avant-plan est donné par

$$F(:, j) = X(:, j) - a_j b,$$

de sorte que la matrice complète s'écrit

$$F = X - ba^\top.$$

Chaque colonne de F représente l'ensemble des pixels dont l'intensité ne peut être expliquée par l'arrière-plan corrigé par l'illumination. Ces valeurs correspondent donc aux objets en mouvement dans la scène, en particulier les piétons.

Nous avons affiché plusieurs colonnes de F sous forme d'images dévectorisées. Sur ces résultats, les piétons apparaissent nettement détachés de l'arrière-plan, tandis que les zones statiques tendent vers zéro. L'extraction des personnes est donc bien réalisée, même si certaines zones peuvent présenter du bruit résiduel lié aux variations d'illumination ou à la présence de textures complexes.

À partir de la succession des colonnes de F, il est également possible de reconstruire une vidéo montrant uniquement l'avant-plan en mouvement. Cette animation met en évidence les trajectoires des différents piétons, et confirme la capacité de la méthode à séparer efficacement mouvement et arrière-plan.

Afin d'illustrer la qualité de l'extraction du mouvement, nous présentons ci-dessous quatre images du foreground prises à différents instants parmi la séquence. Pour comparaison, nous montrons également une image originale extraite de la vidéo.

(a) Avant-plan (image 1)

(b) Avant-plan (image 25)

(c) Avant-plan (image 50)

(d) Avant-plan (image 100)

FIGURE 4.1 – Exemples d'images du foreground obtenues après suppression de l'arrière-plan. Les piétons apparaissent nettement détachés du décor statique.

FIGURE 4.2 – Image originale correspondante.

Ces résultats montrent que la méthode par optimisation linéaire permet d'extraire correctement les personnes en mouvement dans la scène, tandis que l'arrière-plan est fortement atténué. L'animation obtenue en concaténant les 100 images du foreground reproduit les trajectoires des

piétons et illustre clairement la séparation entre mouvement et arrière-plan. L'animation complète du foreground est fournie dans le fichier `foreground.gif`

Chapitre 5

Raffinement itératif de la solution

Une fois les vecteurs a et b déterminés, il est possible d'améliorer l'approximation en alternant les estimations : à partir de a, on recalcule un nouveau b, puis à partir de ce b, on recalcule un nouveau a, et ainsi de suite. Cette procédure, connue sous le nom d'optimisation alternée, vise à diminuer progressivement l'erreur de reconstruction

$$\|X - ba^\top\|_1.$$

À chaque itération, les programmes linéaires permettant de recalculer a et b sont identiques à ceux décrits dans les sections précédentes, mais appliqués successivement. La norme de l'erreur est évaluée à chaque étape afin de vérifier si l'algorithme converge et si la qualité de la reconstruction s'améliore.

5.1 Résultats numériques

Dans notre implémentation, nous avons exécuté quelques itérations de cette stratégie alternée. Nous avons constaté que la quantité $\|X - ba^\top\|_1$ diminue légèrement au cours des premières itérations, ce qui montre que l'approche permet bien d'affiner la reconstruction. Cependant, l'amélioration demeure relativement limitée, ce qui s'explique par le fait que le problème original est très bruité et que la structure de rang 1 imposée à $B = ba^\top$ limite la capacité de modélisation.

5.2 Résultat visuel

D'un point de vue visuel, les changements sont faibles : les silhouettes des piétons restent globalement identiques d'une itération à l'autre, et l'arrière-plan ne se modifie pratiquement pas. L'essentiel du gain se trouve donc dans une légère réduction des artefacts sur certains pixels, mais l'aspect général du foreground demeure similaire.

Ainsi, bien que l'optimisation alternée permette bien de réduire l'erreur $\|X - ba^\top\|_1$, elle n'apporte qu'une amélioration marginale à la qualité visuelle des images extraites.

Annexe A

Code Python pour la suppression de l'arrière-plan

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
from scipy.io import savemat
from scipy.optimize import linprog

data = loadmat('pedsX_il.mat')
X = data['X']
p, t = X.shape
print('Donnees chargees :', p, 'pixels et', t,' images\n')
print("Matrice X :")
print(X)

def afficherimage(image, h = 152, w = 232):
    image = image.reshape([h,w], order='F')
    plt.imshow(image.reshape([h,w]), cmap='gray_r')
    plt.axis('off')
    plt.show()

print('Affichage de la premiere image :')
afficherimage(np.copy(X[:,0]))

b = np.zeros(p)

for i in range(p):
    c = np.zeros(t+2)
    c[2:]=1
    A = np.zeros((2*t,t+2))
    bContrainte = np.zeros(2*t)

    for j in range(t):
        A[2*j,2+j] = -1
        A[2*j,0] = -1
        A[2*j,1] = 1
        bContrainte[2*j] = -X[i,j]
```

```
            A[2*j+1,2+j] = -1
            A[2*j+1,0] = 1
            A[2*j+1,1] = -1
            bContrainte[2*j+1] = X[i,j]

        bounds = [(0,None)]*(t+2)

        model = linprog(c=c,A_ub=A,b_ub=bContrainte,bounds=bounds)
        b[i] = model.x[0] -model.x[1]

print('Affichage du fond reconstruit :')
afficherimage(b)

bMat = np.copy(b)
savemat('b.mat', {'b': bMat})
```

Annexe B

Code Python pour l'estimation de l'illumination

```
import numpy as np
from scipy.optimize import linprog
from scipy.io import loadmat
from scipy.io import savemat
import matplotlib.pyplot as plt

b = loadmat('b.mat')['b'].reshape(-1)
data= loadmat('pedsX_il.mat')
X = data['X']
p,t = X.shape

print("Vecteur b :")
print(b)
print("Matrice X :")
print(X)

def afficherimage(image, h = 152, w = 232):
    image = image.reshape([h,w], order='F')
    plt.imshow(image.reshape([h,w]), cmap='gray_r')
    plt.axis('off')
    plt.show()

print("Affichage de la premiere image :")
image_1 = np.copy(X[:,0])
afficherimage(image_1)

a = np.zeros(t)

for i in range (t):
    c = np.zeros(p+2)
    c[:p]=1
    A = np.zeros((p*2,p+2))
    bContrainte = np.zeros(p*2)

    for j in range (p):
        A[2*j, j] = -1
```

```
            A[2*j, p] = -b[j]
            A[2*j, p+1] = b[j]
            bContrainte[2*j] = -X[j, i]

            A[2*j+1, j] = -1
            A[2*j+1, p] = b[j]
            A[2*j+1, p+1] = -b[j]
            bContrainte[2*j+1] = X[j, i]

        bounds = [(0,None)]*(p+2)

        model = linprog(c=c,A_ub= A, b_ub= bContrainte, bounds = bounds)
        a[i]=model.x[p]-model.x[p+1]

        print(f"a[{i}] = {a[i]}")

aMat = np.copy(a)
savemat('a.mat',{'a' : aMat})
```

Annexe C

Code Python pour retrouver l'avant-plan

```
import numpy as np
from scipy.io import loadmat
from scipy.io import savemat
import matplotlib.pyplot as plt


X = loadmat("pedsX_il.mat")["X"]
a = loadmat("a.mat")["a"].reshape(-1)
b = loadmat("b.mat")["b"].reshape(-1)

p, t = X.shape
F = X - np.outer(b, a)
savemat('F.mat', {'F': F})

print(F)

def afficherimage(image, h = 152, w = 232):
    image = image.reshape([h,w], order='F')
    plt.imshow(image.reshape([h,w]), cmap='gray_r')
    plt.axis('off')
    plt.show()



indices = [0,24,49,99]

for j in indices :
    print(f"Images F(; {j})")
    afficherimage(F[:,j])



import imageio
h, w = 152, 232

frames = []

for j in range(t):
```

```
    frame = F[:, j].reshape((h, w), order='F')
    frame_norm = (frame - frame.min()) / (frame.max() - frame.min() + 1
       e-9)
    frame_uint8 =(frame_norm * 255).astype(np.uint8)

    frame_uint8 = 255- frame_uint8

    frames.append(frame_uint8)
imageio.mimsave("foreground.gif", frames, fps=10)
```

Annexe D

Code Python pour raffiner la solution

```
import numpy as np
from scipy.io import loadmat
from scipy.optimize import linprog
import matplotlib.pyplot as plt


X = loadmat("pedsX_il.mat")["X"]
a = loadmat("a.mat")["a"].reshape(-1)
b = loadmat("b.mat")["b"].reshape(-1)

p,t = X.shape

def compute_a(X,b):
    b = np.zeros(p)

    for i in range (t):
        c = np.zeros(p+2)
        c[:p]=1
        A = np.zeros((p*2,p+2))
        bContrainte = np.zeros(p*2)

        for j in range (p):
            A[2*j, j] = -1
            A[2*j, p] = -b[j]
            A[2*j, p+1] = b[j]
            bContrainte[2*j] = -X[j, i]

            A[2*j+1, j] = -1
            A[2*j+1, p] = b[j]
            A[2*j+1, p+1] = -b[j]
            bContrainte[2*j+1] = X[j, i]

        bounds = [(0,None)]*(p+2)

    model = linprog(c=c,A_ub= A, b_ub= bContrainte, bounds = bounds)
    a[i]=model.x[p]-model.x[p+1]

def cumpute_b(X):
```

```
    for i in range(p):
        c = np.zeros(t+2)
        c[2:]=1
        A = np.zeros((2*t,t+2))
        bContrainte = np.zeros(2*t)

        for j in range(t):
            A[2*j,2+j] = -1
            A[2*j,0] = -1
            A[2*j,1] = 1
            bContrainte[2*j] = -X[i,j]

            A[2*j+1,2+j] = -1
            A[2*j+1,0] = 1
            A[2*j+1,1] = -1
            bContrainte[2*j+1] = X[i,j]

        bounds = [(0,None)]*(t+2)

    model = linprog(c=c,A_ub=A,b_ub=bContrainte,bounds=bounds)
    b[i] = model.x[0] -model.x[1]


def norme_L1(X, b, a):
    return np.sum(np.abs(X - np.outer(b, a)))

errors = []
for i in range(5):
    print(f"Iteration␣{i}")
    b = compute_b(X, a)

    a = compute_a(X, b)

    err = norme_L1(X, b, a)
    errors.append(err)

    print("Erreur␣L1␣=", err)
```