INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

MÉTHODES DIRECTES POUR LA RÉSOLUTION DE SYSTÈMES LINÉAIRES

Méthodes itératives et restauration d'image

Doubli Hoda
Ait Taleb Assia
GM3

Encadré par : Mr TONNOIR ANTOINE

Décembre 2023

Table des matières

1 Introduction

Il existe plusieurs façons d'étudier un système linéaire. Lors du Tp précédent, nous avons vu des méthodes "dites" directes, basées sur une factorisation sous la forme d'un produit de la matrice $A\underline{x} = \underline{b}$. Les méthodes directes sont caractérisées par le fait qu'elles donnent la solution en un nombre fini d'étapes (possiblement grand). Au contraire, une méthode itérative convergera théoriquement en un nombre infini d'itérations. "L'espoir" est donc d'avoir une bonne approximation de la solution en peu d'itérations. Une autre caractéristique des méthodes itératives est qu'elles impliquent principalement des calculs de produit matrice-vecteur. C'est un point essentiel pour pouvoir les implémenter efficacement et réduire considérablement le coût mémoire (en particulier pour les matrices creuses).

L'objectif de ce TP est de présenter comment les méthodes itératives de résolution d'un système linéaire peuvent permettre de résoudre des problèmes de restauration d'images. Plus précisément, on s'intéressera à reconstruire au mieux une image dont un certain nombre de pixels ont été enlevé. C'est une problématique classique de l'inpainting.

Dans un premier temps, nous allons explorer quelques méthodes itératives ainsi que leur implémentation numérique sur Python. Ensuite, nous aborderons un problème de traitement d'image : la restauration.

2 Résolution d'un système linéaire

Dans cette partie, on va s'intéresser à la résolution itérative du système linéaire suivant :

$$(-u_{i-1} + 2u_i - u_{i+1}) + \alpha h^2 u_i = f_i, \ \forall i \in [\![1, N]\!] \tag{1}$$

avec

$$u_0 = u_{N+1} = 0 \text{ et } h = \frac{1}{N+1}$$

Nous prendrons f_i tel que la solution exacte soit $u_i = \cos(2\pi i h)$

2.1 Représentation du problème sous forme matricielle

Pour appliquer les méthodes itératives à la résolution de notre système linéaire, nous allons le représenter sous forme d'une équation matricielle, c'est-à-dire $A\underline{u} = \underline{b}$. Étant donné que nous résolvons un système à N inconnus, nous choisirons des matrices A et $\underline{b}$ de dimensions respectives (N, N) et $(N, 1)$. Une fois que nous aurons déterminé A et $\underline{b}$, nous procéderons à une analyse de A afin de déterminer les méthodes itératives à utiliser.

$$\begin{aligned} A\underline{u} = \underline{b} &\Leftrightarrow (Au)_i = b_i \\ &\Leftrightarrow \sum_{j=1}^{n} A_{i,j} u_j = b_i \\ &\Leftrightarrow A_{i1}u_1 + ... + A_{i(i-1)}u_{i-1} + A_{ii}u_i + A_{i(i+1)}u_{i+1} + ... + A_{in}u_n = b_i \end{aligned} \tag{2}$$

En fusionnant les équations (1) et (2), on constate que le terme invariant par rapport à u dans (1) est f_i, et dans (2), c'est b_i. Par une identification immédiate, nous établissons $b_i = f_i, \ \forall i \in [\![1, N]\!]$. En ayant connaissance de la solution exacte de u_i, nous pourrons ainsi formuler $\underline{b}$ de la manière suivante :

$$b_i = \begin{cases} 2 + \alpha h^2 cos(2\pi h i) - cos(2\pi h(i+1)) & \text{si } i = 1 \\ 2 + \alpha h^2 cos(2\pi h i) - cos(2\pi h(i-1)) & \text{si } i = N \\ 2 + \alpha h^2 cos(2\pi h i) - cos(2\pi h(i-1)) - cos(2\pi h(i+1)) & \text{sinon} \end{cases}$$

En ayant trouvé $\underline{b}$, nous en déduisons :

$$A_{i1}u_1 + ... + A_{i(i-1)}u_{i-1} + A_{ii}u_i + A_{i(i+1)}u_{i+1} + ... + A_{in}u_n = (-u_{i-1} + 2u_i - u_{i+1}) + \alpha h^2 u_i$$

En identifiant terme à terme nous trouvons donc :

$$\begin{aligned} A_{i(i-1)} &= -1, \ \forall i \in [\![2, N]\!] \\ A_{ii} &= 2 + \alpha h^2, \ \forall i \in [\![1, N]\!] \\ A_{i(i+1)} &= -1, \ \forall i \in [\![1, N-1]\!] \\ A_{i,j} &= 0 \text{ sinon} \end{aligned}$$

Nous pouvons ainsi définir notre matrice A :

$$\forall i,j \in [\![1,N]\!], A_{ij} = \begin{cases} 2+\alpha h^2, & \text{si } i=j \\ -1, & \text{si } |i-j|=1 \\ 0, & \text{si} |i-j| \geq 2 \end{cases}$$

On obtient alors :

$$A = \begin{bmatrix} 2+\alpha h^2 & -1 & 0 & \dots & 0 \\ -1 & 2+\alpha h^2 & -1 & \dots & 0 \\ 0 & -1 & 2+\alpha h^2 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & -1 \\ 0 & \dots & 0 & -1 & 2+\alpha h^2 \end{bmatrix} \underline{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix} \underline{b} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix}$$

Tel que :

$$A\underline{u} = \underline{b} \Leftrightarrow (-u_{i-1} + 2u_i - u_{i+1}) + \alpha h^2 u_i = f_i, \ \forall i \in [\![1,N]\!]$$

2.2 Propriétés de la matrice A

Dans cette partie nous allons nous intéresser aux propriétés de notre matrice A. Ceci, nous permettra d'implémenter plus facilement les bonnes méthodes itératives pour la résolution numérique de notre système linéaire.

- **Tridiagonale.** Par définition les seuls éléments différents de 0 dans la matrice A sont la diagonale principale et les diagonales inférieures et supérieures adjacentes à la diagonale principale. Ainsi lorsque $|i-j| \geq 2$, $A_{ij} = 0$ c'est à dire que lorsque nous nous situons situons en dehors des trois diagonales principales les éléments de la matrice sont nuls.
- **Symétrique.** En effet comme nous avons une matrice tridiagonale et que les deux sous-diagonales sont égal à -1 alors $A^T = A$ c'est à dire $\forall i,j \in [\![1,N]\!],\ A_{i,j} = A_{j,i}$
- **Définie positive.** Par définition, une matrice symétrique $A \in M(R)$ est définie positive si et seulement si et vérifie $\underline{x}^t A\underline{x} > 0\ \forall \underline{x} \in \mathbb{R}^N \setminus \{0\}$. Comme A est symétrique, nous démontrerons cette propriété pour affirmer qu'elle est définie postive.

Démonstration : Soit $\underline{x} \in \mathbb{R}^N$

$$\begin{aligned} \underline{x}^t A\underline{x} &= \sum_{i=1}^{N}\sum_{j=1}^{N} (A\underline{x})_i x_i \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} x_i A_{i,j} x_j \end{aligned}$$

Or A est une matrice tridiagonale donc $\forall i,j \in [\![1,N]\!],\ |i-j| \geq 2 \Rightarrow A_{i,j} = 0$

$$\begin{aligned} &= \sum_{i=1}^{N} \sum_{j=max(1,i-1)}^{min(N,i+1)} x_i A_{i,j} x_j \\ &= x_1 A_{1,1} x_1 + x_1 A_{1,2} x_2 + x_N A_{N,N} x_N + x_N A_{N,(N-1)} x_{N-1} + \sum_{i=2}^{N-1}\sum_{j=i-1}^{i+1} x_i A_{i,j} x_j \\ &= (x_1)^2(2+\alpha h^2) - x_1 x_2 + (x_N)^2(2+\alpha h^2) - x_N x_{N-1} + \sum_{i=2}^{N-1} (x_i)^2(2+\alpha h^2) - x_i x_{i+1} - x_{i-1} x_i \\ &= (x_1)^2 + (x_N)^2 + \sum_{i=1}^{N-1} \left((x_i)^2 - 2x_i x_{i+1} + (x_{i+1})^2\right) + \alpha h^2 \sum_{i=1}^{N} (x_i)^2 \\ &= (x_1)^2 + (x_N)^2 + \sum_{i=1}^{N-1} (x_i - x_{i+1})^2 + \alpha h^2 \sum_{i=1}^{N} (x_i)^2 \geq 0 \end{aligned}$$

Nous en déduisons que A est positive. Pour démontrer qu'elle est définie nous allons utiliser l'expression que nous

venons de trouver.

$$\underline{x}^t A\underline{x} = 0 \Rightarrow (x_1)^2 + (x_N)^2 + \sum_{i=1}^{N-1}(x_i - x_{i+1})^2 + \alpha h^2 \sum_{i=1}^{N}(x_i)^2 = 0$$

Commme α est positif alors tous les termes sont positifs donc

$$\Rightarrow \begin{cases} (x_1)^2 = 0 \\ (x_N)^2 = 0 \\ \sum_{i=1}^{N-1}(x_i - x_{i+1})^2 = 0 \\ \alpha h^2 \sum_{i=1}^{N}(x_i)^2 = 0 \end{cases}$$

$$\Rightarrow (x_i)^2 = 0,\ \forall i, j \in [\![1, N]\!]$$

$$\Rightarrow \underline{x} = 0$$

Par conséquent, A est bien définie positive.

Pour conclure, au vu des propriétés de la matrice A plusieurs méthodes itératives peuvent être utilisées comme la méthode du gradient conjugué,la méthode de descente de gradient, la méthode de Gauss-Seidel ou encore la méthode de Jacobi.

2.3 Cas du α <0

Nous avons vu dans le cours que si une matrice est à diagonale strictement dominante alors la méthode de Jacobi et Gauss-Seidel converge. Nous disons d'une matrice $M \in \mathcal{M}_N(\mathbb{R})$ qu'elle est à diagonale strictement dominante si et seulement si :

$$|(M)_{i,i}| > \sum_{j=1,\ j\neq i}^{N} |(M)_{i,j}|\ , \forall i \in [\![1, N]\!]$$

En remplacent M par notre matrice A, nous obtenons pour $N > 2$:

$$|(A)_{i,i}| > \sum_{j=1,\ j\neq i}^{N} |(A)_{i,j}|\ , \forall i \in [\![1, N]\!] \iff |2 + \alpha h^2| > \sum_{j=\max(1,i-1),\ j\neq i}^{\min(N,i+1)} |(A)_{i,j}|\ , \forall i \in [\![1, N]\!]$$

$$\iff \begin{cases} |2 + \alpha h^2| > |A_{1,2}| \\ |2 + \alpha h^2| > |A_{N-1,N}| \\ |2 + \alpha h^2| > |A_{i-1,i}| + |A_{i,i+1}|\ , \forall i \in [\![2, N-1]\!] \end{cases}$$

$$\iff \begin{cases} |2 + \alpha h^2| > 1 \\ |2 + \alpha h^2| > 1 \\ |2 + \alpha h^2| > 2\ , \forall i \in [\![2, N-1]\!] \end{cases}$$

$$\iff |2 + \alpha h^2| > 2$$

$$\iff \alpha > 0 \text{ ou } \alpha < -\frac{4}{h^2}$$

Donc pour $N > 2$ et $\alpha < 0$ si $\alpha < -\frac{4}{h^2}$ alors A est à diagonal strictement dominante par conséquent la méthode de Jacobi et de Gauss-Seidel convergent. Pour $N = 2$, nous obtenons $|(A)_{1,1}| > |(A)_{1,2}|$ et $|(A)_{2,2}| > |(A)_{2,1}|$, donc nous en déduisons que $|2 + \alpha h^2| > 1$ donc si $\alpha < 0$ alors $\alpha < -\frac{3}{h^2}$.

Nous avons également vu que pour $\alpha \geq 0$ la matrice A est définie positive, est-ce le cas pour certains $\alpha < 0$?
Nous savons que $A \in \mathcal{M}_N(\mathbb{R})$ et A est symétrique alors en considérant $A \in \mathcal{M}_N(\mathbb{C})$ nous en déduisons que A est hermitienne ($A = A^*$). Nous pouvons donc utiliser la propriété suivante : A est définie positive $\iff$ les valeurs propres de A sont strictement positives. Comment calculer les valeurs propres de A ?
A est une matrice particulière que l'on appelle les matrices de Toeplitz

$$A = \begin{pmatrix} a_0 & a_{-1} & a_{-2} & \dots & \dots & a_{-n+1} \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \dots & \dots & a_2 & a_1 & a_0 \end{pmatrix}$$

De plus A est tridiagonale donc A est semblable à cette matrice avec $a = 2 + \alpha h^2$ et $b = c = -1$:

$$A = \begin{pmatrix} a & b & 0 & \dots & \dots & 0 \\ c & a & b & \ddots & & \vdots \\ 0 & c & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & b & 0 \\ \vdots & & \ddots & c & a & b \\ 0 & \dots & \dots & 0 & b & a \end{pmatrix}$$

Or pour ces matrices spécifiques les valeurs propres sont données par cette formule $\lambda_k = a + 2\sqrt{bc}\cos(k\pi/(N+1))$ avec $k \in [\![1, N]\!]$. Donc les valeurs propres de A sont $\lambda_k = 2 + \alpha h^2 + 2\cos(k\pi/(N+1))$ avec $k \in [\![1, N]\!]$. Regardons à quelles conditions sur α nous avons $\forall k \in [\![1, N]\!],\ \lambda_k > 0$

$$\begin{aligned} \forall k \in [\![1, N]\!],\ \lambda_k > 0 &\iff \lambda_N > 0 \text{ car } x \mapsto \cos(\pi x) \text{ est décroissant sur } [0, 1] \\ &\iff 2 + \alpha h^2 + 2\cos\left(\frac{N\pi}{N+1}\right) > 0 \\ &\iff \alpha > -2\frac{1 + \cos\left(\frac{N\pi}{N+1}\right)}{h^2} \end{aligned}$$

Par conséquent pour $\alpha > -2\frac{1+\cos\left(\frac{N\pi}{N+1}\right)}{h^2}$, A reste définie positive.

3 Résolution numérique

3.1 Calcul du produit Ax efficacement

Qu'est que le produit $A\underline{x}$? Nous avons défini notre matrice A de sorte à calculer le vecteur suivant si $\underline{x} = \underline{u}$:

$$(A\underline{u})_i = (-u_{i-1} + 2u_i - u_{i+1}) + \alpha h^2 u_i,\ \forall i \in [\![1, N]\!]$$

Nous pourrions itérer de 1 à N pour calculer les termes un par un. Cependant, dans une optique d'efficacité, nous optons pour la vectorisation du programme. En clair, plutôt que de calculer individuellement la somme de chaque composante, nous allons directement effectuer la somme des vecteurs. À noter que bien que la somme des vecteurs est équivalent à la somme des composantes, la bibliothèque NumPy optimise ces calculs en déléguant une partie du programme en langage C.

Pour réécrire notre suite de composante à calculer sous forme de vecteurs nous allons introduire un vecteur $\underline{\tilde{x}}$ de dimension $N + 2$ tel que sa première composante et sa dernière composante soient égales à 0 et $\tilde{x}_i = x_{i-1},\ \forall i \in [\![2, N+1]\!]$. En posant $\underline{x^1}$ qui est égal au N premières composantes de $\underline{\tilde{x}}$, $\underline{x^3}$ qui est égal au N dernières composantes de $\underline{\tilde{x}}$ et $\underline{x^2}$ qui est égal à $\underline{\tilde{x}}$ auquel nous retirons la première et la dernière composante. Nous avons donc :

$$A\underline{x} = -\underline{x^1} + (2 + \alpha h^2)\underline{x^2} - \underline{x^3}$$

Nous implémenterons donc cette formule pour calculer le produit Ax.

3.2 La méthode de Jacobi

La méthode de Jacobi est un algorithme itératif pour résoudre des systèmes linéaires. L'idée principale est d'écrire la matrice du système sous forme de somme d'une matrice diagonale avec une matrice à diagonale nulle et d'utiliser cette décomposition pour itérer l'algorithme. Soit $A \in \mathcal{M}_N(\mathbb{R})$, nous noterons la décomposition de Jacobi par $A = M - N$ avec $M = Diag(A)$ et $N = Diag(A) - A$ et nous exprimerons l'itération sous la forme suivante :

$$\begin{cases} \text{Initialisation} : & x^{(0)} \in \mathbb{R}^n \text{ donné} \\ \text{Itération } k \geq 0 : & x^{(k+1)} = M^{-1}Nx^k + M^{-1}b \end{cases}$$

$\underline{\mathbf{x}} \leftarrow \underline{\mathbf{x}}^0$

$\underline{\mathbf{r}}^0 \leftarrow A\underline{\mathbf{x}} - \underline{\mathbf{b}}$

Tant que $\|\underline{\mathbf{r}}\| \geq \varepsilon \|\underline{\mathbf{r}}^0\|$

$\underline{\mathbf{y}} \leftarrow M^{-1}\underline{\mathbf{r}}$

$\underline{\mathbf{x}} \leftarrow \underline{\mathbf{x}} - \underline{\mathbf{y}}$

$\underline{\mathbf{r}} \leftarrow A\underline{\mathbf{x}} - \underline{\mathbf{b}}$

3.3 Implémentation numérique

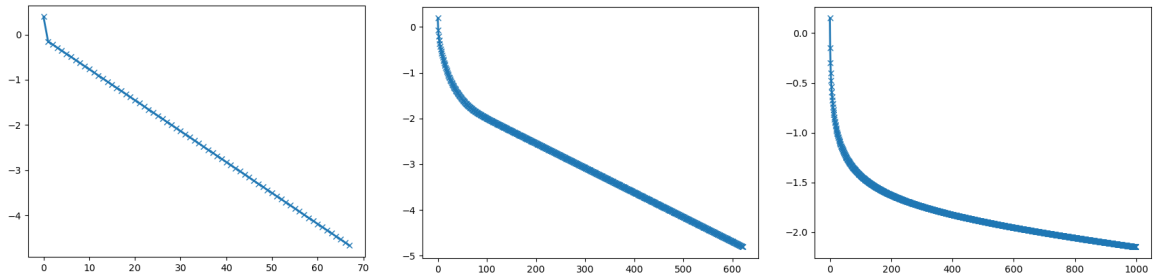
Pour implémenter la méthode de descente de gradient, nous avons deux options. Soit nous adoptons la méthode de descente de gradient avec un pas fixe. Pour optimiser l'algorithme, nous choisirions le pas optimal donné par cette formule $\frac{2}{\lambda_1+\lambda_N}$ avec λ_1 la plus petite valeur propre en module et λ_N la plus grande. Nous pourrions les calculer en remarquant que notre matrice est de la forme d'une matrice de Toeplitz et tridiagonale (voir cas particulier $\alpha < 0$). Cependant, nous préférerons utiliser la deuxième option qui est plus efficace : la descente de gradient à pas variable. Pour la descente de gradient, nous avons appliquer l'algorithme vu en cours.

3.4 Influence de N et α pour les différentes méthodes

Dans cette partie, nous allons afficher à l'aide d'un graphe en lin-log, le résidus trouvés à chaque itération. Pour cela, nous allons exécuter le main.py en testant différentes valeurs de N et α.

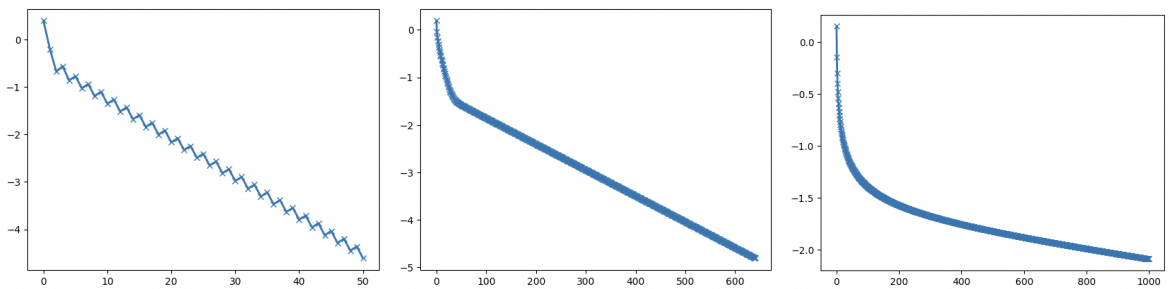
Variations de N avec α fixé :

On va tout d'abord fixer notre α à 1.0.



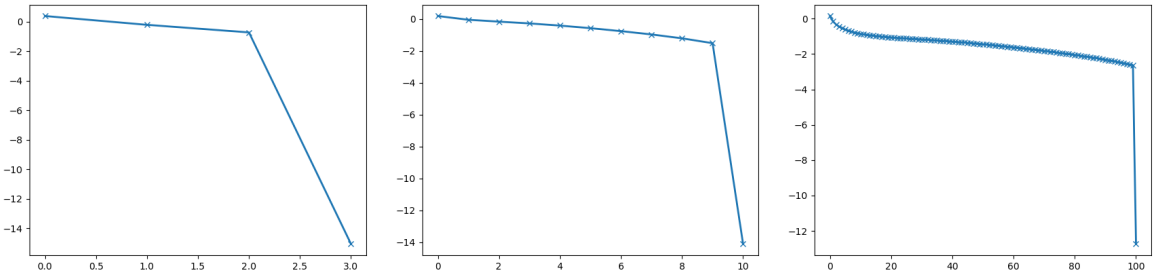
(a) N=5, IteMax = 67/1000 (b) N=20, IteMax = 622/1000 (c) N=200, IteMax = 999/1000

FIGURE 1 – Méthode de Jacobi



(a) N=5, IteMax = 50/1000 (b) N=20, IteMax = 640/1000 (c) N=200, IteMax = 999/1000

FIGURE 2 – Méthode de descente de Gradient



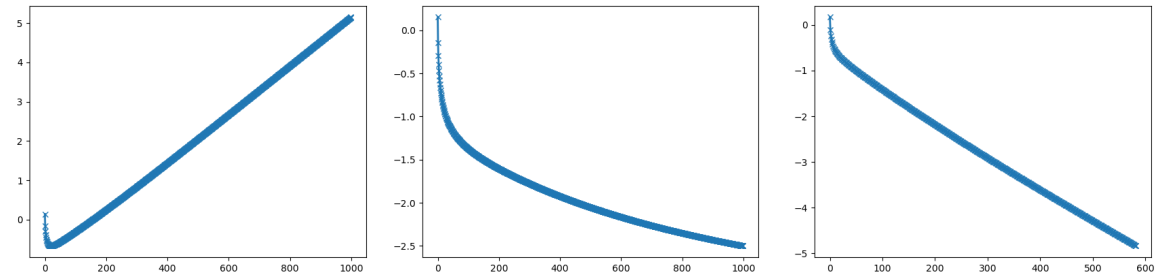
(a) N=5, IteMax = 3/1000 (b) N=20, IteMax = 10/1000 (c) N=200, IteMax = 100/1000

FIGURE 3 – Méthode du Gradient Conjugué

Nous pouvons observer que lorsque le N augmente alors le nombre d'itération augmente également, ce qui peut-être expliqué par l'augmentation de calcul fait par la machine.

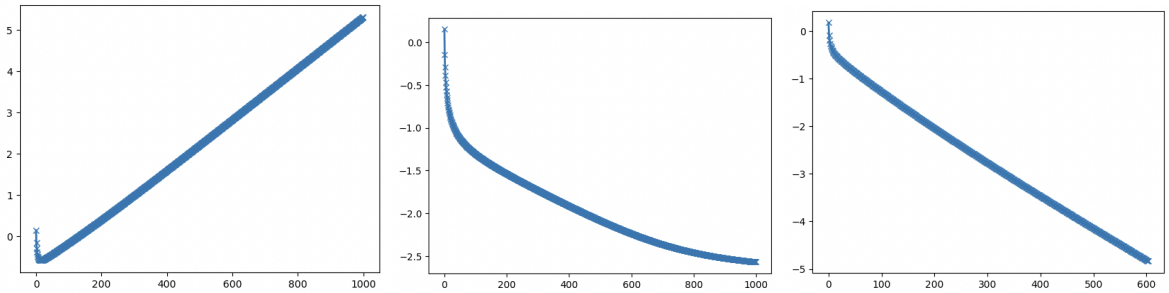
Variations de α avec N fixé :

Fixons maintenant notre N à 100.



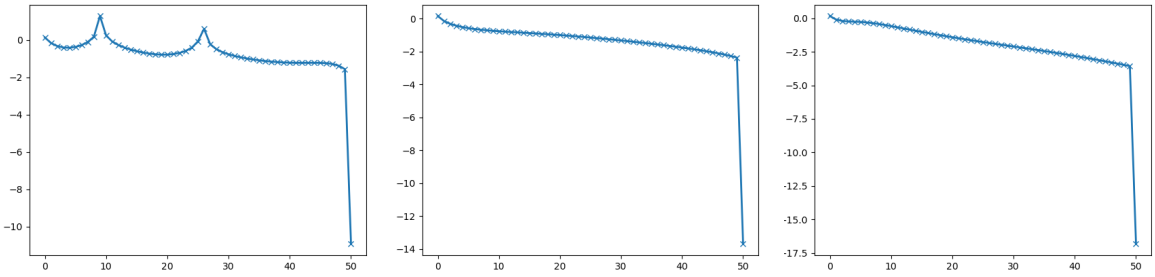
(a) $\alpha = -300$, IteMax = 999/1000 (b) $\alpha = 0$, IteMax = 999/1000 (c) $\alpha = 300$, IteMax = 582/1000

FIGURE 4 – Méthode de Jacobi



(a) $\alpha = -300$, IteMax = 999/1000 (b) $\alpha = 0$, IteMax = 999/1000 (c) $\alpha = 300$, IteMax = 603/1000

FIGURE 5 – Méthode de descente de Gradient



(a) $\alpha = -300$, IteMax = 50/1000 (b) $\alpha = 0$, IteMax = 50/1000 (c) $\alpha = 300$, IteMax = 50/1000

FIGURE 6 – Méthode du Gradient Conjugué

Nous remarquons que la variation du α a un impact sur la vitesse de convergence de notre méthode selon s'il est positif ou négatif.

Pour les α positifs, nous constatons que plus notre α augmente plus les résidus à la dernière itération est petit, c'est-à-dire que la valeur de α va jouer sur la vitesse de convergence. Cela peut être expliqué mathématiquement pour la méthode de Jacobi. Nous savons que la méthode de Jacobi converge si et seulement si la matrice d'itération à un rayon spectral strictement inférieur à 1 de plus la méthode converge rapidement lorsque le rayon spectral est proche de 0. La matrice d'itération est égale à $M^{-1}N$ nous avons donc que la matrice d'itération est égale à $\forall i, j \in [\![1, N]\!]$

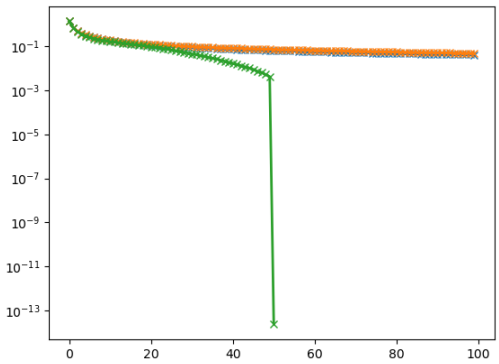
$$(M^{-1}N)_{i,j} = \begin{cases} \frac{1}{2+\alpha h^2} \text{ si } |i-j| = 1 \\ 0 \text{ sinon} \end{cases}$$

Comme nous avons une matrice de Toeplitz tridiagonale, nous pouvons calculer les valeurs propres et donc le rayon spectral, les valeurs propres sont données par cette formule : $\forall k \in [\![1, N]\!]$

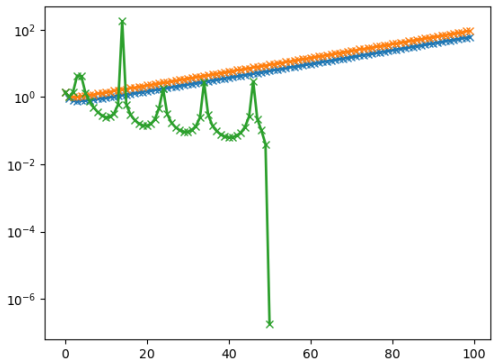
$$\lambda_k = \left(\frac{1}{2+\alpha h^2}\right)^2 \cos\left(\frac{k\pi}{1+N}\right)$$

Nous en déduisons que le rayon spectral de la matrice d'itération (pour $\alpha > 0$) est égal à $\left(\frac{1}{2+\alpha h^2}\right)^2 \cos\left(\frac{\pi}{1+N}\right)$. Donc lorsque α augmente le rayon spectral diminue par conséquent la méthode converge plus rapidement.

Nous remarquons également à travers ces différents graphiques que la méthode de Gradient et Jacobi ont une vitesse de convergence semblable contrairement à la méthode du Gradient Conjugué qui est beaucoup plus rapide.



(a) Graphe des résidus pour N=100 et α = 0,1



(b) Graphe des résidus pour N=100 et α = -1000

Pour les α positifs, les méthodes de Jacobi et Descente de gradient vont converger plus lentement. Tandis que la méthode de gradient conjugué va converger beaucoup plus rapidement. Une fois que la solution a été trouvé,le programme va s'arrêter.

Ensuite, pour les α négatifs, à partir d'un certain rang ($\alpha = \frac{-4}{h^2}$), Jacobi et Descente de gradient divergent clairement. Quand à la méthode de gradient conjugué, on voit des "sauts" sur le graphe des résidus, qui correspondent à des moments de divergence, mais la méthode finit par converger lorsque la solution a était trouvée.

4 Restauration d'images

Imaginons une image I en nuances de gris composée de Nx×Ny pixels. Nous représentons cette image I à l'aide d'une matrice de taille $N_x \times N_y$. Supposons que I subisse des détériorations causées par une fonction M (matrice de même taille que I), où $M_{i,j} = 0$ si le pixel (i,j) est supprimé et 1 sinon. L'image détériorée est alors $I^{alt} = M \times I$ (ici, $I^{alt} = M_{i,j} \times I_{i,j}$. Notre objectif est de reconstruire une image I^r qui se rapproche au mieux de l'image d'origine. Pour cela, nous cherchons à résoudre le problème de minimisation suivant : Trouver I^r tel que $J(I^r)$ soit minimum où,

$$J(I^r) = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}(M_{i,j}I^r_{i,j} - M_{i,j}I^{alt}_{i,j})^2 + R\sum_{i=2}^{N_x-1}\sum_{j=2}^{N_y-1}(4I^r_{i,j} - I^r_{i-1,j} - I^r_{i,j-1} - I^r_{i+1,j} - I^r_{i,j+1})^2$$

FIGURE 8 – Image détérioré à gauche et image restauré à droite

Minimiser $J(I^r)$ en gardant cette expression, n'est pas simple à faire car on dispose d'une relation qui est indicée en 2 dimensions.
Ainsi, nous allons donc simplifier l'expression en procédant à un changement de variable qui transformera les matrice $M, I^r et I^{alt}$ de dimension $N_x \times N_y$ en un vecteur avec $N_x N_y$ composantes.

4.1 Simplification du problème

Les vecteur $\underline{I^r et M}$ ne sont pas encore défini et il n'y a pas de façon naturelle de les définir. En effet,$\underline{I^r}$ est un vecteur (unicolonne) contenant $N_x N_y$ éléments alors que la notation "géométrique" $I^r_{i,j}$ indique une matrice. Nous devons donc numéroter le double indice (i, j) par un monoindice l. Nous prenons ici la relation : $l = (i-1)N_y + j$.
Le vecteur $\underline{I^r}$ des inconnues est donc donné par la colonne suivante :

$$I^r = \begin{bmatrix} I^r_{11} \\ I^r_{12} \\ \vdots \\ I^r_{1N_y} \\ I^r_{21} \\ \vdots \\ I^r_{2N_y} \\ \vdots \\ I^r_{N_x 1} \\ \vdots \\ I^r_{N_x N_y} \end{bmatrix}$$

Pour simplifier les notations, on note $N = N_x N_y$.
On a les correspondances suivantes :

$$\begin{cases} (i,j), & \Leftrightarrow l \\ (i,j+1), & \Leftrightarrow l+1 \\ (i,j-1), & \Leftrightarrow l-1 \\ (i+1,j), & \Leftrightarrow l+N_y \\ (i-1,j), & \Leftrightarrow l-N_y \end{cases}$$

Dans cette partie nous allons essayer d'écrire

$$J(I^r) = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\left(M_{i,j}I^r_{i,j} - M_{i,j}I^{\text{alt}}_{i,j}\right)^2 + \mathcal{R}\sum_{i=2}^{N_x-1}\sum_{j=2}^{N_y-1}\left(4I^r_{i,j} - I^r_{i-1,j} - I^r_{i,j-1} - I^r_{i+1,j} - I^r_{i,j+1}\right)^2$$

sous la forme :

$$\| A\tilde{I}^r - b \|_2^2 \text{ en posant } I^r_{i,j} = \tilde{I}^r_l \text{ avec } l = (i-1)N_y + j$$

Nous séparerons notre problème en 2 sous-problèmes, nous commencerons par trouver A_1 de dimension (N, N) et b_1 de dimension $(N, 1)$ de sorte à avoir :

$$\| A_1\tilde{I}^r - b_1 \|_2^2 = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\left(M_{i,j}I^r_{i,j} - M_{i,j}I^{\text{alt}}_{i,j}\right)^2$$

Puis nous trouverons A_2 de dimension (N, N) et b_2 de dimension $(N, 1)$ de sorte à avoir :

$$\| A_2\tilde{I}^r - b_2 \|_2^2 = \mathcal{R}\sum_{i=2}^{N_x-1}\sum_{j=2}^{N_y-1}\left(4I^r_{i,j} - I^r_{i-1,j} - I^r_{i,j-1} - I^r_{i+1,j} - I^r_{i,j+1}\right)^2$$

En trouvant A_1, b_1, A_2, b_2 nous trouverons A de dimension $(2N, N)$ et b de dimesnion $(2N, 1)$ en les écrivant sous forme de matrice bloc :

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix},\ b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Commençons par trouver A_1 et b_1 :

$$\begin{aligned}
\| A_1\tilde{I}^r - b_1 \|_2^2 &= \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\left(M_{i,j}I^r_{i,j} - M_{i,j}I^{\text{alt}}_{i,j}\right)^2 \\
\sum_{l=1}^{N}\left(A_1\tilde{I}^r - b_1\right)_l^2 &= \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\left(M_{i,j}I^r_{i,j} - M_{i,j}I^{\text{alt}}_{i,j}\right)^2 \\
\sum_{l=1}^{N}\left(\left(A_1\tilde{I}^r\right)_l - (b_1)_l\right)^2 &= \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\left(M_{i,j}I^r_{i,j} - M_{i,j}I^{\text{alt}}_{i,j}\right)^2 \\
\sum_{l=1}^{N}\left(\sum_{k=1}^{N}(A_1)_{l,k}\tilde{I}^r_k - (b_1)_l\right)^2 &= \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\left(M_{i,j}I^r_{i,j} - M_{i,j}I^{\text{alt}}_{i,j}\right)^2
\end{aligned}$$

Pour identifier plus facilement b_1 nous pouvons annuler I^r et $\tilde{I}^r$, nous avons donc

$$\sum_{l=1}^{N} (b_1)_l^2 = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} \left(M_{i,j} I_{i,j}^{\text{alt}}\right)^2$$

Nous obtenons donc

$$\forall l \in [\![1, N]\!],\ (b_1)_l = M_{i,j} I_{i,j}^{\text{alt}} \text{ avec } l = N_y(i-1)+j,\ i \in [\![1, N_x]\!],\ j \in [\![1, N_y]\!]$$

Maintenant pour trouver A_1 nous allons utiliser le même procédé, nous annulons b_1 pour identifier plus facilement les termes

$$\sum_{l=1}^{N} \left(\sum_{k=1}^{N} (A_1)_{l,k} \tilde{I}_k^r \right)^2 = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} \left(M_{i,j} I_{i,j}^r\right)^2$$

$$\sum_{l=1}^{N} \left((A_1)_{l,l} \tilde{I}_l^r + \sum_{k=1, k\neq l}^{N} (A_1)_{l,k} \tilde{I}_k^r \right)^2 = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y} \left(M_{i,j} I_{i,j}^r\right)^2$$

Nous obtenons donc

$$\forall l \in [\![1, N]\!],\ (A_1)_{l,l} = M_{i,j} \text{ avec } l = N_y(i-1)+j,\ i \in [\![1, N_x]\!],\ j \in [\![1, N_y]\!] \text{ et } (A_1)_{l,k} = 0,\ \forall k \in [\![1, N]\!],\ k \neq l$$

Finalement, $\forall l, k \in [\![1, N]\!]$

$$(A_1)_{l,k} = \begin{cases} M_{i,j},\ \text{si } l = k \text{ avec } l = N_y(i-1)+j,\ i \in [\![1, N_x]\!],\ j \in [\![1, N_y]\!] \\ 0,\ \text{sinon} \end{cases}$$

$$(b_1)_l = M_{i,j} I_{i,j}^{\text{alt}} \text{ avec } l = N_y(i-1)+j,\ i \in [\![1, N_x]\!],\ j \in [\![1, N_y]\!]$$

Pour trouver A_2 et b_2 nous utiliserons le même raisonnement. Dans un premier temps nous constatons que si nous annulons I^r alors $\mathcal{R}\sum_{i=2}^{N_x-1}\sum_{j=2}^{N_y-1}\left(4I_{i,j}^r - I_{i-1,j}^r - I_{i,j-1}^r - I_{i+1,j}^r - I_{i,j+1}^r\right)^2 = 0$ donc $||b_2|| = 0$ par conséquent $b_2 = 0$

Pour calculer A_2 nous allons identifier terme à terme comme pour A_1

$$\sum_{l=1}^{N} \left(\sum_{k=1}^{N} (A_2)_{l,k} \tilde{I}_k^r \right)^2 = \mathcal{R} \sum_{i=2}^{N_x-1}\sum_{j=2}^{N_y-1} \left(4I_{i,j}^r - I_{i-1,j}^r - I_{i,j-1}^r - I_{i+1,j}^r - I_{i,j+1}^r\right)^2$$

Comme nous avons $\forall i \in [\![2, N_X - 1]\!],\ \forall j \in [\![2, N_y - 1]\!]$ avec $l = N_y(i-1)+j$

$$\begin{aligned} I_{i,j}^r &= \tilde{I}_l^r \\ I_{i-1,j}^r &= \tilde{I}_{l-N_y}^r \\ I_{i+1,j}^r &= \tilde{I}_{l+N_y}^r \\ I_{i,j-1}^r &= \tilde{I}_{l-1}^r \\ I_{i,j+1}^r &= \tilde{I}_{l+1}^r \end{aligned}$$

Nous en déduisons donc $\forall l, k \in [\![1, N]\!]$ avec $l = N_y(i-1)+j$

$$(A_2)_{l,k} = \begin{cases} 4\sqrt{\mathcal{R}},\ \text{si } l = k,\ i \neq 1,\ i \neq N_x\ j \neq 1,\ j \neq N_y \\ -\sqrt{\mathcal{R}},\ \text{si } (l - N_y = k \text{ ou } l + N_y = k \text{ ou } l - 1 = k \text{ ou } l + 1 = k) \text{ et } i \neq 1,\ i \neq N_x\ j \neq 1,\ j \neq N_y \\ 0,\ \text{sinon} \end{cases}$$

De A_1, A_2, b_1 et b_2, nous en déduisons A et b, $\forall l \in [\![1, 2N]\!],\ \forall k \in [\![1, N]\!]$,

$$A_{l,k} = \begin{cases} (A_1)_{l,k},\ \text{si } l \leq N \\ (A_2)_{l-N,k},\ \text{sinon} \end{cases} \qquad b_l = \begin{cases} (b_1)_l,\ \text{si } l \leq N \\ (b_2)_{l-N},\ \text{sinon} \end{cases}$$

Après correction du Tp une nouvelle fonction à minimiser est à étudier. De la même manière que ci-dessus nous allons réécrire :

$$J(I^r) = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\left(M_{i,j}I^r_{i,j} - M_{i,j}I^{\text{alt}}_{i,j}\right)^2 + \mathcal{R}\sum_{i=2}^{N_x-1}\sum_{j=2}^{N_y-1}\left(I^r_{i,j} - I^r_{i+1,j}\right)^2 + \left(I^r_{i,j} - I^r_{i,j+1}\right)^2$$

sous la forme :

$$\parallel A\tilde{I^r} - b \parallel_2^2 \text{ en posant } I^r_{i,j} = \tilde{I}^r_l \text{ avec } l = (i-1)N_y + j$$

Cette fois-ci, nous séparerons notre problème en 3 sous-problèmes, de sorte à avoir $A1$, $A2$, $A3$ de dimension de (N, N) et b_1, b_2, b_3 de dimension $(N, 1)$ tel que

$$\parallel A_1\tilde{I^r} - b_1 \parallel_2^2 = \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\left(M_{i,j}I^r_{i,j} - M_{i,j}I^{\text{alt}}_{i,j}\right)^2$$

$$\parallel A_2\tilde{I^r} - b_2 \parallel_2^2 = \mathcal{R}\sum_{i=2}^{N_x-1}\sum_{j=2}^{N_y-1}\left(I^r_{i,j} - I^r_{i+1,j}\right)^2$$

$$\parallel A_3\tilde{I^r} - b_3 \parallel_2^2 = \mathcal{R}\sum_{i=2}^{N_x-1}\sum_{j=2}^{N_y-1}\left(I^r_{i,j} - I^r_{i,j+1}\right)^2$$

En trouvant A_1, b_1, A_2, b_2, A_3, b_3 nous trouverons A de dimension $(3N, N)$ et b de dimension $(3N, 1)$ en les écrivant sous forme de matrice bloc :

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix},\ b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Nous trouvons A_1, A_2, A_3, b_1, b_2 et b_3 de la même manière que pour la précédente formule. Nous remarquons que le A_1 et le b_1 sont exactement les mêmes et pour trouver A_2, b_2, A_3 et b_3. Nous utiliserons un raisonnement similaire que pour le A_2 et b_2 de la précédente formule, par identification nous trouvons $\forall l, k \in [\![1, N]\!]$ avec $l = N_y(i-1) + j$:

$$(A_2)_{l,k} = \begin{cases} \sqrt{\mathcal{R}},\ \text{si } l = k,\ i \neq 1,\ i \neq N_x\ j \neq 1,\ j \neq N_y \\ -\sqrt{\mathcal{R}},\ \text{si } l + N_y = k,\ i \neq 1,\ i \neq N_x\ j \neq 1,\ j \neq N_y \\ 0,\ \text{sinon} \end{cases},\ b_2 = 0$$

$$(A_3)_{l,k} = \begin{cases} \sqrt{\mathcal{R}},\ \text{si } l = k,\ i \neq 1,\ i \neq N_x\ j \neq 1,\ j \neq N_y \\ -\sqrt{\mathcal{R}},\ \text{si } l + 1 = k,\ i \neq 1,\ i \neq N_x\ j \neq 1,\ j \neq N_y \\ 0,\ \text{sinon} \end{cases},\ b_3 = 0$$

De A_1, A_2, A_3, b_1, b_2 et b_3 nous en déduisons A et b, $\forall l \in [\![1, 3N]\!],\ \forall k \in [\![1, N]\!]$,

$$A_{l,k} = \begin{cases} (A_1)_{l,k},\ \text{si } l \leq N \\ (A_2)_{l-N,k},\ \text{si } N+1 \geq l \leq 2N \\ (A_3)_{l-2N,k},\ \text{sinon} \end{cases} \quad b_l = \begin{cases} (b_1)_l,\ \text{si } l \leq N \\ (b_2)_{l-N},\ \text{si } N+1 \geq l \leq 2N \\ (b_3)_{l-2N},\ \text{sinon} \end{cases}$$

(q.2)

On a donc montré que le problème de minimisation revient à trouver $\tilde{I^r} \in \mathbb{R}^N$ minimisant $\left\| A\tilde{I^r} - \underline{b} \right\|_2^2$ (a).
Nous pouvons simplifier encore ce problème en montrant que minimiser (a) revient à résoudre l'équation suivante : $A^t A\underline{\tilde{I^r}} = A^t\underline{b}$.

Pour commencer, on sait que $\|\underline{v}\|_2^2 = \underline{v^t v}$ ainsi, en développant l'expression (a), on va se retrouver avec une forme plus simple pour calculer le gradient. (dire pk on calcule le gradient)

$$\begin{aligned} \left\| A\underline{\tilde{I^r}} - \underline{b} \right\|_2^2 &= (A\underline{\tilde{I^r}} - \underline{b})^t(A\underline{\tilde{I^r}} - \underline{b}) \\ &= ((A\underline{\tilde{I^r}})^t - \underline{b}^t)(A\underline{\tilde{I^r}} - \underline{b}) \\ &= (A\underline{\tilde{I^r}})^t(A\underline{\tilde{I^r}}) - (A\underline{\tilde{I^r}})^t\underline{b} - \underline{b}^t(A\underline{\tilde{I^r}}) + \underline{b}^t\underline{b} \end{aligned} \tag{3}$$

On se retrouve donc avec une forme plus simple qui va nous permettre de calculer le gradient de f par rapport à I^r où $f(I^r) = \left\| A\underline{\tilde{I^r}} - \underline{b} \right\|_2^2$

$$\begin{aligned} \nabla f(I^r) &= \frac{\partial((A\tilde{I^r})^t(A\tilde{I^r}) - (A\tilde{I^r})^t\underline{b} - \underline{b}^t(A\tilde{I^r}) + \underline{b}^t\underline{b})}{\partial\tilde{I^r}} \\ &= \frac{\partial(A\tilde{I^r})^t(A\tilde{I^r})}{\partial\tilde{I^r}} - \frac{\partial(A\tilde{I^r})^t\underline{b}}{\partial\tilde{I^r}} - \frac{\partial\underline{b}^t(A\tilde{I^r})}{\partial\tilde{I^r}} + \frac{\partial(\underline{b}^t\underline{b})}{\partial\tilde{I^r}} \end{aligned} \tag{4}$$

Il nous faut maintenant calculer les 4 dérivées partielles qu'on obtient.

$\frac{\partial(\underline{b}^t\underline{b})}{\partial\tilde{I^r}} = 0$ car $(\underline{b}^t\underline{b})$ ne dépend pas de I^r.

Ensuite, l'expression b^tAI^r peut être décomposé en une somme de termes étant de la forme $b_iA_{ij}I^r_j$. Quand on prend la dérivée partielle par rapport à I^r_k, seule la k-ème composante apparait dans la somme ainsi, la dérivée partielle par rapport à I^r_k de chaque terme $b_iA_{ij}I^r_j$ est b_iA_{ij} si j = k et 0 sinon.
La somme de tous ces termes est donnée par :

$$\begin{aligned} \frac{\partial(\underline{b}^t(AI^r)}{\partial\tilde{I^r_k}} &= \sum_{i=1}^{N} b_iA_{ik} \\ &= b_1A_{1k} + \cdots + b_NA_{Nk} \\ &= A_{k1}b_1 + \cdots + A_{kN}b_N \\ &= \sum_{i=1}^{N} A_{ki}b_i \\ &= \sum_{i=1}^{N} A^t_{ik}b_i \\ &= A^t\underline{b} \end{aligned} \tag{5}$$

Considérons l'expression $(AI^r)^T\underline{b}$, où $A^T\underline{b}$ est une matrice colonne.

La dérivée partielle par rapport à I^r est donnée par :

$$\frac{\partial(AI^r)^T\underline{b}}{\partial I^r} = \begin{bmatrix} \frac{\partial({I^r}^TA^T\underline{b})}{\partial I^r_1} \\ \frac{\partial({I^r}^TA^T\underline{b})}{\partial I^r_2} \\ \vdots \\ \frac{\partial({I^r}^TA^T\underline{b})}{\partial I^r_N} \end{bmatrix}$$

Chaque composante de cette matrice colonne est obtenue en prenant la dérivée partielle correspondante par rapport à la composante correspondante de I^r, et ainsi :

$$\frac{\partial(AI^r)^T\underline{b}}{\partial I^r} = A^T\underline{b}$$

Passons maintenant au calul de la dérivée partielle suivante :

$$\frac{\partial((A\tilde{I^r})^t(A\tilde{I^r})}{\tilde{I^r}}$$

. Elle est de la forme u*v ainsi on peut appliquer la formule de la dérivée d'un produit $(u * v)' = u'v + uv'$ au calcule de la dérivée partielle.

$$
\begin{aligned}
\frac{\partial((A\tilde{I^r})^t(A\tilde{I^r})}{\tilde{I^r}_k} &= \frac{\partial((A\tilde{I^r})^t}{\partial\tilde{I^r}_k}(A\tilde{I^r}) + (A\tilde{I^r})^t\frac{\partial((A\tilde{I^r})}{\partial\tilde{I^r}_k} \\
&= \frac{\partial}{\partial\tilde{I}^r_k}\sum_{i=1}^{N}(\tilde{I^r}_i A_i^t)(A\tilde{I^r}) + (A\tilde{I^r})^t\frac{\partial}{\partial\tilde{I^r}_k}\sum_{i=1}^{N}A_i\tilde{I}_i^r \\
&= \sum_{i=1}^{N}\frac{\partial}{\partial\tilde{I}^r_k}\tilde{I}_i^r A_i^t(A\tilde{I^r}) + (A\tilde{I^r})^t\sum_{i=1}^{N}\frac{\partial}{\partial\tilde{I^r}_k}I_i^r A_i^t \\
&= \sum_{i=1}^{N}\delta_{ik}A_i^t(A\tilde{I^r}) + (A\tilde{I^r})^t\sum_{i=1}^{N}\delta_{ik}A_i \\
&= A_k^t(A\tilde{I^r}) + (A\tilde{I^r})^t A_k
\end{aligned}
\tag{6}
$$

La dérivée partiel de ce produit par rapport à la composante $\tilde{I^r}_k$ vaut $A_k^t(A\tilde{I^r}) + (A\tilde{I^r})^t A_k$.
Le 1èr terme correspond à la multiplication de la ligne k de A^t avec le produit $(A\tilde{I^r})$. Le 2ème terme quand à lui est la multiplication du vecteur ligne $(A\tilde{I^r})^t$ par rapport à la colonne k de A. Or la multiplication du second terme revient à multiplier là kème colonne de A^t avec le vecteur colonne $(A\tilde{I^r})$.

On en conclut que :

$$\frac{\partial((A\tilde{I^r})^t(A\tilde{I^r})}{\tilde{I^r}} = A^t(A\tilde{I^r}) + A^t(A\tilde{I^r})$$

Ainsi en regroupant toutes ces dérivée partielle calculé, on obtient :

$$\nabla f(I^r) = A^t(A\tilde{I^r}) + A^t(A\tilde{I^r}) - A^t\underline{b} - A^t\underline{b} = 2A^t(A\tilde{I^r}) - 2A^t\underline{b}$$

Ainsi, minimiser $f(\tilde{I^r})$ revient à résoudre l'équation :

$$
\begin{aligned}
\nabla f(I^r) = 0 &\Leftrightarrow 2A^t(A\tilde{I^r}) - 2A^t\underline{b} = 0 \\
&\Leftrightarrow A^t(A\tilde{I^r}) = A^t\underline{b}
\end{aligned}
\tag{7}
$$

4.2 Pour aller plus loin

$\forall k \in [\![1, N]\!]$

$$
\begin{aligned}
\underline{\tilde{y}} = A^tA\underline{\tilde{I}^r} &\iff \tilde{y}_k = \left(A^tA\underline{\tilde{I}^r}\right)_k \\
&\iff \tilde{y}_k = \sum_{k'=1}^{3N}(A^tA)_{k,k'}\left(\underline{\tilde{I}^r}\right)_{k'} \\
&\iff \tilde{y}_k = \sum_{k'=1}^{3N}\left(\sum_{l=1}^{3N}(A^t)_{k,l}(A)_{l,k'}\right)\left(\underline{\tilde{I}^r}\right)_{k'} \\
&\iff \tilde{y}_k = \sum_{k'=1}^{3N}\left(\sum_{l=1}^{3N}(A)_{l,k}(A)_{l,k'}\right)\left(\underline{\tilde{I}^r}\right)_{k'} \\
&\iff \tilde{y}_k = \sum_{k'=1}^{3N}\left(\sum_{l=1}^{N}(A_1)_{l,k}(A_1)_{l,k'} + \sum_{l=1}^{N}(A_2)_{l,k}(A_2)_{l,k'} + \sum_{l=1}^{N}(A_3)_{l,k}(A_3)_{l,k'}\right)\left(\underline{\tilde{I}^r}\right)_{k'} \\
&\iff \tilde{y}_k = M_{i,j}I^r_{i,j} + \sum_{k'=1}^{3N}\left(\sum_{l=1}^{N}(A_2)_{l,k}(A_2)_{l,k'} + \sum_{l=1}^{N}(A_3)_{l,k}(A_3)_{l,k'}\right)\left(\underline{\tilde{I}^r}\right)_{k'}, \text{ avec } k = N_y(i-1)+j \\
&\iff \tilde{y}_k = M_{i,j}I^r_{i,j} + \sum_{k'=1}^{3N}\left(\sum_{l=1}^{N}(A_2)_{l,k}(A_2)_{l,k'} + \sum_{l=1}^{N}(A_3)_{l,k}(A_3)_{l,k'}\right)\left(\underline{\tilde{I}^r}\right)_{k'}, \text{ avec } k = N_y(i-1)+j \\
&\iff \tilde{y}_k = \begin{cases} M_{i,j}I^r_{i,j} \text{ si } i = 1 \text{ ou } i = N_x \text{ ou } j = 1 \text{ ou } j = N_y \\ M_{i,j}I^r_{i,j} + \sum_{k'=1}^{3N}\left((A_2)_{k-N_y,k}(A_2)_{k-N_y,k'} + (A_2)_{k,k}(A_2)_{k,k'} + \sum_{l=1}^{N}(A_3)_{l,k}(A_3)_{l,k'}\right)\left(\underline{\tilde{I}^r}\right)_{k'} \text{ sinon} \end{cases},
\end{aligned}
$$

Cette question n'a pas été aboutie.

4.3 Résolution numérique

Le but de cette partie, va être de restaurer une image détérioré en utilisant les méthodes itératives suivante : Jacobi, descente de gradient, gradient conjugué.
On a montrer dans la partie précédente, que reconstruire au mieux une image détérioré (ici l'image est une matrice de pixels), revient à résoudre l'équation suivante : $A^t(A\tilde{I^r}) = A^t\underline{b}$.

(q.pour aller plus loin)

(q.Codes)

On vient de montrer que le résultat du produit matrice-vecteur $y_l = A^t(A\tilde{I^r})$ est équivalent à

$$y_l = M_{ij}I^r_{ij} + R(4I^r_{ij} - I^r_{i-1,j} - I^r_{i,j-1} - I^r_{i+1,j} - I^r_{i,j+1})$$

On a donc besoin d'une fonction qui va permettre de calculer ce vecteur y_l.
Nous allons tout d'abord introduire une autre fonction que l'on appelle "laplace" qui calcule le terme de régularisation de la fonction $J(I^r)$. Il corespond à :

$$\sum_{i=2}^{N_x-1}\sum_{j=2}^{N_y-1}(4I^r_{i,j} - I^r_{i-1,j} - I^r_{i,j-1} - I^r_{i+1,j} - I^r_{i,j+1})$$

Avant de chercher la façon la plus optimal d'écrire la fonction laplace, on a tout simplement garder cette forme matricielle et utilisé une double boucle pour effectuer nos calculs. On a donc obtenue ce pseudo-code :

Algorithm 1 Calcul du terme de régularisation version matricielle

```
1:  Fonction LAPLACE(img)
2:      imgres = 4*img
3:      [Nx,Ny]= taille de img
4:      Pour i allant de 1 à N_x − 1 Faire
5:          Pour j allant de 1 à N_y − 1 Faire
6:              imgres[i,j] = imgres[i,j] - img[i-1,j] - img[i,j-1] - img[i+1,j] - img[i,j+1]
7:          Fin Pour
8:      Fin Pour
9:      retourner imgres
10: Fin Fonction
```

On a vite compris qu'il fallait absolument optimiser ce code car si on voulait restaurer une image avec une méthode itératives au choix, chaque itérations durait environ 2 min.
Ici, l'utilisation de la double boucles n'est pas du tout adapté pour de grandes matrices. Et justement la matrice utilisé est de taille 4032×3024. Chaque opérations prend donc un certain temps. Ainsi, nous en sommes arrivé à supprimer les 2 boucles et à diviser le temps d'éxecution par 30. Ce pseudo-code utilise des opérations de slicing qui est une méthode permettant d'extraire une sous-section d'un tableau.

Algorithm 2 Calcul du terme de régularisation version matricielle (slicing)

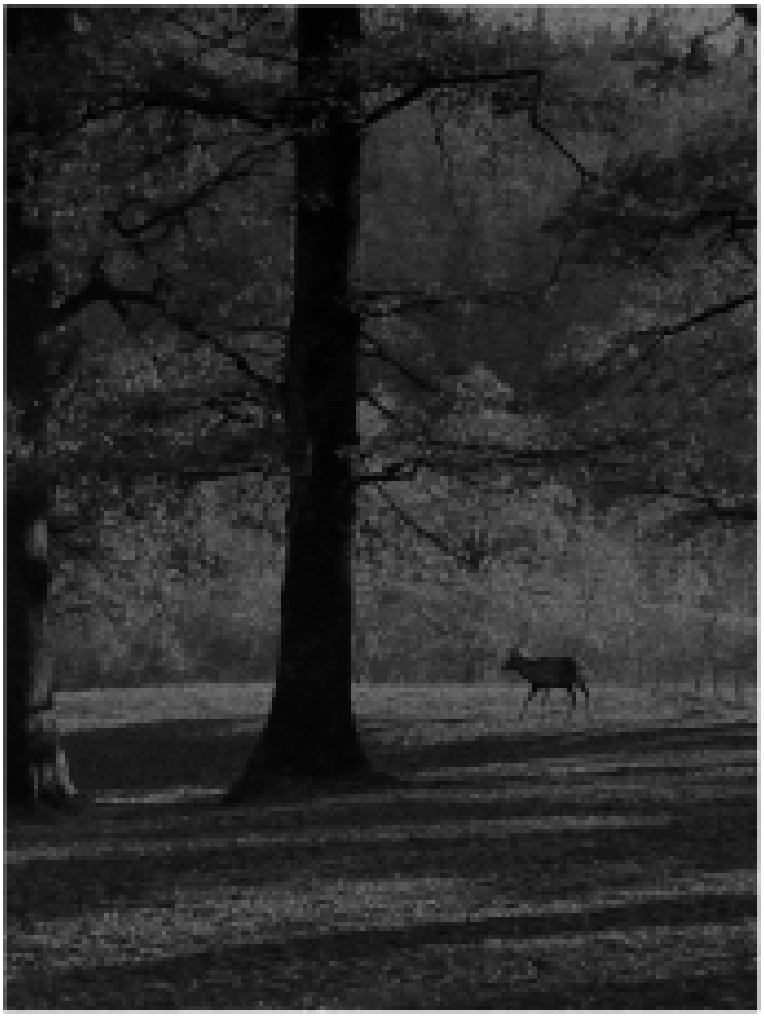
```
1: Fonction LAPLACE(img)
2:     imgres = 4*img
3:     imgres[1 :-1,1 :-1] -= img[ :-2,1 :-1] + img[1 :-1, :-2] + img[2 :,1 :-1] + img[1 :-1,2 :]
4:     retourner imgres
5: Fin Fonction
```

Après avoir calculer notre termes de régularisation, on utilise une autre fonction noté produitImg qui utilise la fonction laplace et calcule ainsi notre vecteur y_l

Algorithm 3 Calcul du produit vecteur y_l

```
1: Fonction produitImg(x,M,R,Nx,Ny)
2:     xxImg = x.reshape(Nx,Ny)
3:     newImg = M*xxImg
4:     newImg = newImg + R*laplace(xxImg)
5:     retourner newImg.reshape(-1)
6: Fin Fonction
```

Pour permettre la restauration d'une image, nous avons à notre disposition une matrice M de taille $N_x * N_y$ où les coefficients sont de 0 pour chaque pixel enlevé et 1 sinon, ainsi que de toutes les données nécessaire.
Voici l'image que l'on doit restaurer :



Pour cela, nous avons testé les 3 méthodes itératives, Jacobi, Descente de Gradient et Gradient Conjugué et on remarque que certaines méthode ne fonctionne pas pour tout les paramètres et que les temps d'éxecution ne sont pas les mêmes.

Méthode de Gradient conjugué :

En effet, pour la méthode de gradient conjugué, si nous mettons un R (le paramètre de régularisation) pas très grand entre 0 et 20 exclus, l'image final est net. Tandis qu'un grand R (nous avons testé R=30), va rendre l'image très floue. De plus, pour n'importe quel nombre d'itérations l'image va être très net. Pour 100 itérations, l'image est net au bout de la 55ème itération. En effet, cela est dûe au fait que la méthode de gradient conjugué fonctionne avec un nombre petite d'itérations et qu'elle converge rapidement.
Concernant le temps d'exécution, le temps passé par itération est le plus long comparé aux 2 autres méthode utilisé.

Méthode de Descente de Gradient :

Cette méthode fonctionne aussi mais contrairement à la méthode de gradient conjugué, pour un grand R, l'image va

ressortir beaucoup moins bien que celle restauré en utilisant la méthode de gradient conjugué. Pour un petit nombre d'itération (par exemple 10), l'image ressort avec des imperfections est très sombre.

Méthode de Jacobi :

Comme la méthode de gradient conjugué, l'image est net pour un R pas trop grand (inférieur à 30). En effet en fixant R à 1, ou 0.1, on obtient une image net. Cette méthode a aussi ses inconvénients : elle converge lentement. Donc on ne doit pas fixer le nombre d'itérations trop petit car sinon le rendu de l'image ne sera pas assez net.
Nous avons remarqué que cette méthode est la plus rapide concernant le temps d'éxecution. Cela est dûe au fait que la méthode de Jacobi est facile à exécuter et nécessite peu de calcul par rapport aux autres méthodes.

En conclusion, pour toutes ces méthodes, nous obtenons une image net comme ci-dessous (avec le bon choix des paramètres).

5 Conclusion

On a remarqué que pour utiliser les méthodes itératives, en générale, il n'est pas nécessaire de connaître A ou stocker A, seulement de pouvoir calculer les produits $A\underline{x}$ à l'aide des valeurs connues de la matrice A.

En conclusion, différentes méthodes itératives telles que la descente de gradient, le gradient conjugué, et la méthode de Jacobi sont des outils puissants dans le domaine de la résolution de systèmes linéaires.
Chacune de ces méthodes a ses avantages et ses limites, et le choix approprié dépend souvent des caractéristiques spécifiques du problème.
La descente de gradient est polyvalente mais peut être sensible au choix du pas, le gradient conjugué est particulièrement efficace pour les matrices symétriques définies positives, tandis que la méthode de Jacobi peut être adaptée à des systèmes linéaires moins complexe. En considérant la structure de la matrice et les propriétés du problème, nous pouvons sélectionner la méthode la mieux adaptée pour atteindre une convergence rapide et efficace vers la solution optimale.