TP Lissages

O. Wilk

Cnam - CSC110

TP 2016 - 2017

Les Travaux Pratiques (TP) de l'Unité d'Enseignement CSC110 du Cnam Paris consistent à :

- (1) - introduire mathématiquement les différentes méthodes mises en jeu au sein de ce TP,
- (2) - souligner leurs principaux résultats théoriques,
- (3) - expliquer la mise en oeuvre numérique imposée,
- (4) - programmer en python (sous l'environnement Jupyter) ce qui est demandé,
- (5) - exécuter et commenter les applications numériques demandées.

L'ensemble de ce travail (les points (1) à (5)) doit être rédigé sous la forme d'un compte rendu. Il est fortement conseillé de commencer la rédaction (des points (1) à (3)) dès le début des travaux pratiques et de suivre le contenu de l'énoncé.

Introduction

Nous allons mettre en oeuvre trois méthodes de lissage (f image à lisser, u image solution du problème de lissage, Ω domaine de l'image) :

- La première méthode consiste à minimiser le critère suivant ($\varepsilon_1 \in \mathbb{R}^+$):

$$J1(u) = \frac{1}{2} \int_\Omega |f-u|^2 d\Omega + \frac{\varepsilon_1}{2} \int_\Omega |\nabla u|^2 d\Omega. \tag{1}$$

  Il s'agit du lissage linéaire (isotrope).

- La deuxième méthode consiste à minimiser le critère suivant ($\varepsilon_2, a \in \mathbb{R}^+$) :

$$J2(u) = \frac{1}{2} \int_\Omega |f-u|^2 d\Omega + \frac{\varepsilon_2}{2} \int_\Omega \sqrt{1 + a * |\nabla u|^2} d\Omega. \tag{2}$$

  Il s'agit d'un lissage non linéaire (anisotrope).

- La troisième méthode consiste à minimiser le critère suivant ($\varepsilon_3, g \in \mathbb{R}^+$) :

$$J2(u) = \frac{1}{2} \int_\Omega |f-u|^2 d\Omega + g \int_\Omega |\nabla(u-f)| d\Omega + \frac{\varepsilon_3}{2} \int_\Omega |\nabla u|^2 d\Omega. \tag{3}$$

  Il s'agit d'un lissage non linéaire (tixotrope).

1 Lissage linéaire

La méthode de lissage linéaire consiste à minimiser le critère $J1$ (1). Ce problème de minimisation est équivalent au problème variationnel suivant :

$$\forall v \in H_0^1(\Omega), \int_\Omega uv \, d\Omega + \varepsilon_1 \int_\Omega \nabla u.\nabla v \, d\Omega = \int_\Omega fv \, d\Omega. \tag{4}$$

(**EXPLIQUER**)

C'est aussi le problème faible du problème fort suivant (ν normale extérieure au bord du domaine Ω) :

$$\begin{cases} -\varepsilon_1 \Delta u + u = f \text{ dans } \Omega, \\ \\ \dfrac{\partial u}{\partial \nu} = 0 \text{ sur } \partial\Omega. \end{cases} \tag{5}$$

(**EXPLIQUER**)

Nous allons utiliser cette dernière forme pour résoudre notre problème. La méthode des différences finies nous permet d'approximer l'opérateur Laplacien Δ de la manière suivante (cf. Figure 1) :

$$\begin{cases} \Delta u = \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} \\ \\ \text{avec } \dfrac{\partial^2 u_{i,j}}{\partial x^2} \simeq \dfrac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\delta x)^2} \\ \\ \text{et } \dfrac{\partial^2 u_{i,j}}{\partial y^2} \simeq \dfrac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\delta y)^2}. \end{cases} \tag{6}$$

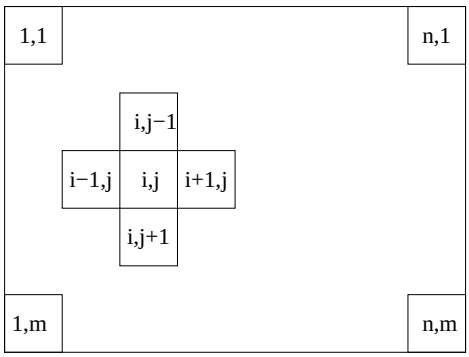

Figure 1: Pixels dans une image.

Ce qui nous donne l'approximation suivante en prenant $\delta x = \delta y = h$ ($u \in \mathbb{R}^{n \times m}$):

$$(\Delta u)_{(i,j)} \simeq \frac{u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i,j-1} + u_{i-1,j}}{h^2}. \tag{7}$$

Le système d'équations (i, j) s'écrit :

$$\begin{cases} \ldots \\ -\varepsilon_1 \dfrac{u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i,j-1} + u_{i-1,j}}{h^2} + u_{i,j} = f_{i,j} \\ \ldots \end{cases} \tag{8}$$

Et il faut évidemment appliquer la condition limite de Neumann homogène.
(**EXPLIQUER**)

1.1 Résolution par une méthode itérative : la méthode de Jacobi

Il s'agit d'effectuer la résolution du système linéaire de type $Au = b$ par la méthode de Jacobi (processus itératif permettant d'éviter de stocker la matrice) :

$$u_{i,j}^{p+1} = \frac{1}{A_{(i,j)(i,j)}}(b_{i,j} - \sum_{(k,l)=(1,1),(k,l)\neq(i,j)}^{(n,m)} A_{(i,j)(k,l)} u_{k,l}^{p}), \text{ notation vue en cours} \quad (9)$$

avec i et j variant respectivement de 1 à n et de 1 à m. A ce stade, il faut bien sur prendre en compte la condition de Neumann homogène.
(**EXPLIQUER**)

(**PROGRAMMATION 1 : IL S'AGIT ICI DE REPRENDRE À SON COMPTE LE PROGRAMME DONNÉ EN COURS/ED, DE L'EXPLIQUER ET DE LE MODIFIER POUR INTÉGRER LA CONDITION DE NEUMANN HOMOGÈNE (version boucle uniquement)**)

(**PROGRAMMATION 2 : AJOUTER À L'OUTIL "CALCUL DE LA FONCTIONNELLE $J1$" PERMETTANT DE SUIVRE LE BON FONCTIONNEMENT DE LA MÉTHODE, L'OUTIL "CALCUL DE LA NORME L_2 DU RÉSIDU $b - Au$" ET VISUALISER LA COURBE DE CETTE NORME EN FONCTION DES ITÉRATIONS**)

(**APPLICATION NUMÉRIQUE : REFAIRE L'APPLICATION NUMÉRIQUE ASSOCIÉE AU PROGRAMME DONNÉ EN COURS/ED EN Y AJOUTANT LA COURBE NORME DU RÉSIDU EN FONCTION DES ITÉRATIONS POUR $(h, \varepsilon_1) = (1, 1)$ ET $(0.1, 100)$), EVALUER LA DIFFÉRENCE ENTRE LES DEUX IMAGES, EXPLIQUER**)

1.2 Résolution par une méthode directe

Dans le cas d'une méthode de résolution directe, il est utile de stocker la matrice en utilisant le rangement vectoriel pour les images (c.a.d. : k = (i-1)*m + j). L'image f est rangée de manière vectorielle dans vf. Pour la matrice A associée, nous pouvons écrire :

$$\begin{cases} A(k, k+m) = -\varepsilon_1/h^2, \\ A(k, k+1) = -\varepsilon_1/h^2, \\ A(k, k) = 1 + \varepsilon_1/h^2, \\ A(k, k-1) = -\varepsilon_1/h^2, \\ A(k, k-m) = -\varepsilon_1/h^2. \end{cases} \quad (10)$$

Nous décomposons la matrice A de la manière suivante : $A = M - \varepsilon_1 K$.
(**EXPLIQUER**)

(**PROGRAMMATION 3 : POUR LE STOCKAGE, NOUS UTILISONS LE SOUS PACKAGE "SPARSE" DE SCIPY :**

```
import numpy as np
import scipy.sparse as sp_sp

nsom = n*m
M = sp_sp.eye(nsom) # matrice identité

# Les trois lignes suivantes permettent de créer une matrice "morse" dont chaque ligne
# est composée des composantes de K (seules les composantes non nulles sont stockées) :
e = np.ones(nsom) # vecteur composé uniquement de 1 de dimension nsom
K = sp_sp.spdiags([e,e,-4*e,e,e],[-m,-1,0,1,m],nsom, nsom)
K = K.tocsc()

# Corrections pour la prise en compte de la condition limite de Neumann homogène :
#     la composante (k,l) de K est accessible/modifiable de la manière suivante :
#     K[k,l] = la valeur adéquate,
```

```
#     procéder d'abord en gérant les bords sans les coins puis les coins
#     utiliser la figure 1 pour vous aider.
...
```

APRÈS AVOIR MODIFIÉE LA MATRICE K POUR IMPLÉMENTER LA CONDITION LIMITE DE NEUMANN, NOUS POUVONS RÉSOUDRE DE LA MANIÈRE SUIVANTE ("vu" contient la solution) :

```
import scipy.sparse.linalg as sp_sp_lin

sp_sp_lin.use_solver(useUmfpack = True)  # False pour superLU

A = M - eps*K
A = A.tocsc() # changement de format en vue de la résolution

solver = sp_sp_lin.factorized(A) # factorisation
vu = solver(vf) # résolution
```

POUR VISUALISER LA SOLUTION À L'AIDE D'UNE "COLORMAP" DE TONS DE GRIS, VOUS POUVEZ UTILISER LES LIGNES SUIVANTES :

```
import matplotlib.pyplot as plt

plt.imshow(vu.reshape([n,m]), cmap=plt.cm.gray)
plt.title("la solution")
plt.show()
```

)

(**APPLICATIONS NUMÉRIQUES : REFAIRE L'APPLICATION NUMÉRIQUE DE LA SECTION PRÉCÉDENTE PUIS COMPARER LES DEUX MÉTHODES DE RÉSOLUTION PROGRAMMÉES.**)

2 Lissage non linéaire (anisotrope)

Pour ce travail, nous allons minimiser $J2(u)$. Pour cela, nous devons calculer sa dérivée directionnelle. Avant, généralisons la forme de cette fonctionnelle en nous intéressant à :

$$J2_b(u) = \frac{1}{2}\int_\Omega (f-u)^2 d\Omega + \frac{\varepsilon_2}{2}\int_\Omega \varphi(|\nabla u|^2) d\Omega. \tag{11}$$

Calculer la dérivée directionnelle de $J2_b$ et lorsque celle-ci s'annule, retrouver le problème fort suivant :

$$-\varepsilon_2 div(\varphi^{'}(|\nabla u|^2)\nabla u) + u = f \text{ dans } \Omega. \tag{12}$$

(**EXPLIQUER**)

Il est possible de réécrire l'équation différentielle pour mieux apprécier l'intérêt d'un tel problème. Ainsi nous pouvons obtenir :

$$-\varepsilon_2[\varphi^{'}(|\nabla u|^2)\Delta u + 2*\varphi^{''}(|\nabla u|^2)|\nabla u|^2 \frac{\nabla u^T}{|\nabla u|}\begin{pmatrix} \partial_x^2 u & \partial_{xy} u \\ \partial_{xy} u & \partial_y^2 u \end{pmatrix}\frac{\nabla u}{|\nabla u|}] + u = f \tag{13}$$

(**EXPLIQUER**)

Revenons au problème initial avec $\varphi(t) = \sqrt{1+a*t}$ et $t = |\nabla u|^2$:

$$\varphi^{'}(t) = \frac{a}{2*\sqrt{1+a*t}}, \varphi^{''}(t) = -\frac{1}{2}\varphi^{'}(t)\frac{a}{1+a*t}. \tag{14}$$

Dans ce cas, étudions un peu : $\frac{\nabla u^T}{|\nabla u|}\begin{pmatrix} \partial_x^2 u & \partial_{xy} u \\ \partial_{xy} u & \partial_y^2 u \end{pmatrix}\frac{\nabla u}{|\nabla u|}$.

$\frac{\nabla u}{|\nabla u|}$ est un vecteur de norme 1 indiquant la direction que prend le gradient au point considéré. En un point où nous avons un gradient ∇u non nul, nous décidons de nommer la direction de notre gradient "1". En ce même point, la direction "2" indique la direction orthogonale au gradient. Nous avons donc par rapport à cette base locale :

$$\frac{\nabla u}{|\nabla u|} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Cela nous permet d'écrire :

$$-\varepsilon_2\, \varphi^{'}(|\nabla u|^2)\, \partial_2^2 u - \varepsilon_2\, \varphi^{'}(|\nabla u|^2)\, \{\, 1 \, - \, \frac{a * |\nabla u|^2}{1 + a * |\nabla u|^2}\, \}\, \partial_1^2 u + u = f \tag{15}$$

(**EXPLIQUER**)

Il ne nous reste plus qu'à étudier le comportement de $\varphi^{'}(t)$ et de $\varphi^{'}(t)\{\, 1 \, - \, \frac{a*t}{1+a*t}\, \}$ en fonction de $t = |\nabla u|^2$.

- Si $t = 0$, nous avons : $\varphi^{'}(t) = a$ et $1 \, - \, \frac{a*t}{1+a*t} = 1$.
- Si $t \to \infty$, nous avons : $1 \, - \, \frac{a*t}{1+a*t} \to 0$. Ainsi bien que $\varphi^{'}(t)$ tend aussi vers 0, il tend vers 0 moins vite que $\varphi^{'}(t)\, \{\, 1 \, - \, \frac{a*t}{1+a*t}\, \}$. En particulier cela signifie que lorsque le gradient est grand, le lissage s'effectue plus dans la direction orthogonale du gradient de u que suivant la direction du gradient (cf. Figure 2).

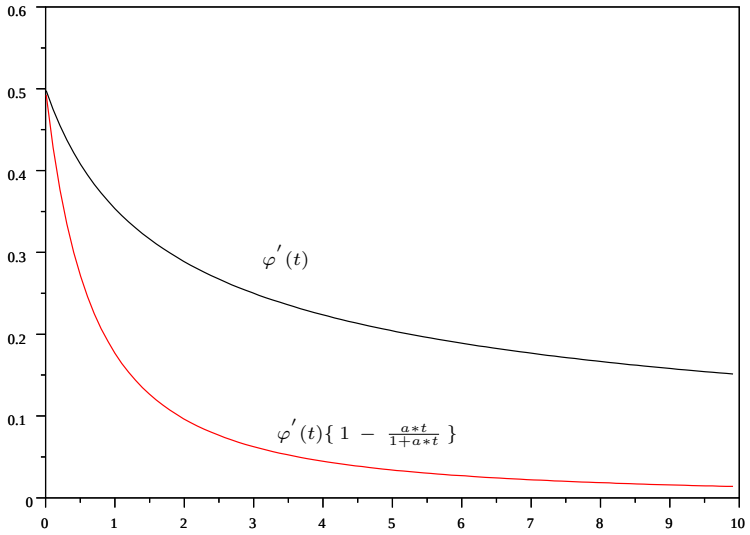

Figure 2: Différence d'évolution entre $\varphi^{'}(t)$ et $\varphi^{'}(t)\{\, 1 \, - \, \frac{a*t}{1+a*t}\, \}$ pour $a = 1$.

Etant donné le caractère non linéaire de l'opérateur du problème (12), nous utilisons le processus itératif suivant :

$$u^{k+1} = u^k - \rho\, G^k \text{ avec } \rho > 0 \text{ (et pas trop grand !) .} \tag{16}$$

Les calculs précédents nous donnent comme gradient :

$$G = \frac{\partial J2_b}{\partial u} \quad = u - f - \varepsilon_2\, \varphi^{'}(|\nabla u|^2)\, \Delta u - 2\, \varepsilon_2\, \varphi^{''}(|\nabla u|^2)\, \nabla u^T \begin{pmatrix} \partial_x^2 u & \partial_{xy} u \\ \partial_{xy} u & \partial_y^2 u \end{pmatrix} \nabla u \tag{17}$$

ou encore:

$$\begin{aligned} G = \frac{\partial J2_b}{\partial u} \quad &= u - f - \varepsilon_2\, \varphi^{'}(|\nabla u|^2)\, [\partial_x^2 u + \partial_y^2 u] \\ &-2\, \varepsilon_2\, \varphi^{''}(|\nabla u|^2)\, [\partial_x^2 u (\partial_x u)^2 + 2 \partial_{xy} u \partial_x u \partial_y u + \partial_y^2 u (\partial_y u)^2] \end{aligned} \tag{18}$$

Pour calculer les différentes dérivées au point (i, j), nous utilisons les approximations par différences finies suivantes :

- $\partial_x u_{i,j} = \dfrac{u_{i+1,j} - u_{i-1,j}}{2h}, \partial_y u_{i,j} = \dfrac{u_{i,j+1} - u_{i,j-1}}{2h},$
- $\partial_x^2 u_{i,j} = \dfrac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}, \partial_x^2 u_{i,j} = \dfrac{u_{i,j-1} - 2u_{i,j} + u_{i,j-1}}{h^2},$
- $\partial_{12} u = \dfrac{u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}}{4h^2}.$

Nous choisissons : $a = 10^N / t_{max}$. avec $t_{max} = max(|\nabla u|^2)$. Le code python utile pour obtenir le maximum d'une image (ou d'un vecteur) est :

```
# Pour une image f, la composante maximale de f peut être obtenue de la manière suivante :
f_max = f.max()
```

Nous relions le coefficient de lissage non linéaire à celui du lissage linéaire de la manière suivante :

$$\varepsilon_2 = \frac{2\,\varepsilon_1\,t_{max}}{10^n}.$$

(EXPLIQUER)

(PROGRAMMATION 4 : APRÈS AVOIR IMPORTÉ LES PACKAGES, APRÈS AVOIR RÉCUPÉRÉ L'IMAGE "f" À LISSER COMME POUR LE LISSAGE LINÉAIRE ($f = f * 1.$ POUR ASSURER QUE LES VALEURS DE L'IMAGE f SOIENT CODÉES PAR DES RÉELS) , APRÈS AVOIR DIMENSIONNÉ LES DIFFÉRENTS PARAMÈTRES DE LA MÉTHODE DE LISSAGE NON LINÉAIRE, VOUS DEVEZ UTILISEZ POUR CALCULER LE GRADIENT (18) CE QUI SUIT :

```
def d11(u,h):
    """calcul de la dérivée seconde suivant la direction 1"""
    K = (1./(h**2))*np.array([[0,0,0],[1,-2,1],[0,0,0]])
    return nd.filters.convolve(u,K)
def d22(u,h):
    """calcul de la dérivée seconde suivant la direction 2"""
    K = (1./(h**2))*np.array([[0,1,0],[0,-2,0],[0,1,0]])
    return nd.filters.convolve(u,K)
def d12(u,h):
    """calcul de la dérivée croisée"""
    K = (1./(4*h**2))*np.array([[1,0,-1],[0,0,0],[-1,0,1]])
    return nd.filters.convolve(u,K)
def d1(u,h):
    """calcul de la dérivée première suivant la direction 1"""
    K = (1./(2*h))*np.array([[0,0,0],[-1,0,1],[0,0,0]])
    return nd.filters.convolve(u,K)
def d2(u,h):
    """calcul de la dérivée première suivant la direction 2"""
    K = (1./(2*h))*np.array([[0,-1,0],[0,0,0],[0,1,0]])
    return nd.filters.convolve(u,K)
```

POUR EFFECTUER LES PRODUITS TERME À TERME INTERVENANT DANS (18), PAR EXEMPLE POUR :

$\partial_x^2 u (\partial_x u)^2$

NOUS UTILISONS :

```
d11u = d11(u,h)
d1u = d1(u,h)
operateur = d11u*(d1u)**2
```

AINSI LA VARIABLE "operateur" CONTIENT UNE PARTIE DU DERNIER TERME ENTRE CROCHETS DE (18). POUR METTRE EN OEUVRE LA METHODE, IL FAUT CALCULER LE GRADIENT (18). VOUS POUVEZ UTILISER LA FONCTION SUIVANTE POUR CELA (EN LA COMPLÉTANT CORRECTEMENT) :

```
def grad_lissage_non_lin(u,f,a,eps_2,h):

    d1u = d1(u,h)
    d2u = d2(u,h)
```

```
    d11u = d11(u,h)
    d22u = d22(u,h)
    d12u = d12(u,h)

    laplacien_u = ...

    d1u2 = d1u**2
    d2u2 = d2u**2

    norme_du_2 = d1u2 + d2u2 # norme au carré du gradient de u

    operateur = d11u*d1u2 + ...

    phi_prim = ...
    phi_primprim = ...

    G = u - f - eps_2*phi_prim*lapu - 2*eps_2*phi_primprim*operateur

    return G, norme_du_2
```

SUR CETTE BASE, PROGRAMMER L'ALGORITHME ITÉRATIF ASSOCIÉ À (16), CALCULER $J2$ ET LA NORME L2 DE "G" À CHAQUE ITÉRATION ET STOCKER RESPECTIVEMENT LEURS VALEURS AU SEIN DES VECTEURS "VJ" ET "VG". VOUS VISUALISEREZ VOS RÉSULTATS À L'AIDE DE LA FONCTION SUIVANTE :

```
def visu(f,u,VJ,VG,comm=""):
    plt.figure(figsize=(16,16))
    plt.subplot(2,2,1)
    plt.imshow(f, cmap=plt.cm.gray)
    plt.title("f")
    plt.subplot(2,2,2)
    plt.imshow(u, cmap=plt.cm.gray)
    plt.title(comm)
    plt.subplot(4,1,3)
    plt.plot(VJ)
    plt.title("J")
    plt.subplot(4,1,4)
    plt.plot(VG)
    plt.title("G")
    plt.show()
```

)

(APPLICATION NUMÉRIQUE : VOUS LISSEREZ DE MANIÈRE NON LINÉAIRE L'IMAGE "CERVEAU-IRM-550.PNG" POUR UN NOMBRE D'ITÉRATIONS MAXIMALES DE "200" ASSOCIÉ À L'ALGORITHME ITÉRATIF, POUR $h = 1, \varepsilon_1 = 10, 50, N = 3, 5$ ET POUR $\rho = 1.e - 2$. POUR LES CAS AVEC LA PLUS PETITE VALEUR POUR ε_1, VOUS DEVEZ VOIR LES VALEURS DE LA FONCTIONNELLE $J2$ ET DU GRADIENT ASSOCIÉ DIMINUER RÉGULIÈREMENT. POUR $\varepsilon_1 = 50$, LES COURBES PEUVENT ËTRE UN PEU PLUS CHAHUTÉES. AINSI POUR CES DERNIERS CAS, VOUS POUVEZ DIMINUER ρ PAR DEUX TOUT EN EN MULTIPLIANT LE NOMBRE D'ITÉRATIONS PAR DEUX. LA CONVERGENCE DEVRAIT ÊTRE AMÉLIORÉE.
PRÉSENTER LES 4 IMAGES RÉSULTATS DU LISSAGE NON LINÉAIRE SOUS FORME D'UN TABLEAU (ε_1, N), À L'AIDE DE :

```
plt.figure(figsize=(24,24))
plt.subplot(221)
plt.imshow(u1, cmap=plt.cm.gray) # u1 solution pour eps_1 = 10, N = 3
plt.subplot(222)
plt.imshow(u2, cmap=plt.cm.gray) # u2 solution pour eps_1 = 10, N = 5
plt.subplot(223)
plt.imshow(u3, cmap=plt.cm.gray) # u3 solution pour eps_1 = 50, N = 3
plt.subplot(224)
plt.imshow(u4, cmap=plt.cm.gray) # u4 solution pour eps_1 = 50, N = 5
plt.show()
```

EXPLIQUER LES RÉSULTATS.)

Remarque 1 : Historiquement, le premier filtre de lissage non linéaire à caractère diffusif fut introduit par P. Perona et J. Malik (Scale space and edge detection using anisotropic diffusion - Proc IEEE Comp. Soc. Workshop on Computer Vision (Miami Beach, Nov. 30-Dec. 2, 1987), IEEE Computer Society Press, Washington, 16-22, 1987.) with :

$$g(|\nabla u|^2) = \varphi^{'}(|\nabla u|^2) = \frac{1}{1+|\nabla u|^2/\lambda}, \lambda > 0, \tag{19}$$

associé au modèle de l'équation de la chaleur :

$$\partial_t u = div(g(|\nabla u|^2)\nabla u). \tag{20}$$

La méthode de résolution itérative choisie plus haut permet de faire le lien entre les deux modèles (en discrétisant en temps l'équation précédente). Le modèle que nous traitons est semblable au problème de P. Charbonnier, L. Blanc-Féraud, G. Aubert et M. Barlaud (Two deterministic half-quadratic regularization algorithms for computed imaging - Proc. IEEE Int. Conf. Image Processing (ICIP Austin, Nov 13-16, 1994), Vol. 2, IEEE Computer Society Press, Los Alamitos, 168-172, 1994).

Remarque 2 : Etudions un peu en monodimensionnel le problème (20). Nous pouvons écrire :

$$\partial_t u = \partial_x(g(|\partial_x u|^2)\partial_x u) = \partial_x(\Phi(\partial_x u)) = \Phi^{'}(\partial_x u)\partial_x^2 u, \text{ avec } \Phi(\partial_x u) = g(|\partial_x u|^2)\partial_x u. \tag{21}$$

Dans le cas du modèle de Perona-Malik, on peut s'apercevoir que le signe de $\Phi^{'}(\partial_x u)$ peut changer de signe et en particulier devenir négatif lorsque le gradient devient trop grand ($> \lambda$). Dans ce cas, la résolution de l'équation de la chaleur inverse est un problème instable. Ce n'est pas théoriquement le cas pour le modèle traité dans ce TP.

Travail optionnel : Mettre en place la détermination du ρ optimal.

3 Lissage non linéaire (tixotrope - différences finies)

Le problème consiste à minimiser ($\varepsilon > 0$, $g > 0$, $\mu > 0, \mu \to 0$):

$$J^{\varepsilon,g,\mu}(v) = \frac{\varepsilon}{2}\int_\Omega |\nabla v|^2 + g\int_\Omega \sqrt{\mu + |\nabla(v-f)|^2} + \frac{1}{2}\int_\Omega |v-f|^2, \tag{22}$$

version régularisée de :

$$J^{\varepsilon,g}(v) = \frac{\varepsilon}{2}\int_\Omega |\nabla v|^2 + g\int_\Omega |\nabla(v-f)| + \frac{1}{2}\int_\Omega |v-f|^2, \tag{23}$$

en utilisant l'algorithme suivant :

- Initialisation : $\lambda^0 = 0$
- Etape courante : λ^n connu

$$\begin{cases} -\varepsilon\Delta u + u = f + g\ div(\lambda) \text{ dans } \Omega \text{ avec } \dfrac{\partial u}{\partial \nu} = 0 \text{ sur } \partial\Omega, \\ \lambda^{n+1} = P_B(\lambda^n + \rho\nabla(u^n - f)), 0 < \rho < 2\dfrac{\varepsilon}{g}. \end{cases} \tag{24}$$

avec P_B projection sur la boule unité.

3.1 Travail à faire

- Démontrer le théorème :
  Théorème Si $f \in H_0^1(\Omega)$ t.q. : $||\nabla f||_{0,\infty,\Omega} \leq \dfrac{g}{\varepsilon} \Rightarrow u^{\varepsilon,g} = f$.
- Démontrer la convergence de l'algorithme (en revenant à la formulation variationnelle associée à la formulation forte),
- souligner en quoi la factorisation (méthode directe) est intéressante (par rapport à une méthode itérative),
- expliquer et utiliser le programme ci-dessous avec :
  (cas 0 => eps = 0.01, gc = 0.4), (cas 1 => eps = 0.01, gc = 0.1),
- commenter les résultats.

3.2 Code "ipython notebook"

3.2.1 Importation des packages “python” utiles à notre application

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt  # pour les aspects graphiques

import sys
import scipy.ndimage as nd
import scipy.sparse as sp_sp
import scipy.sparse.linalg as sp_sp_lin
import numpy as np
import time

# option utilisée pour la factorisation d'une des matrices
sp_sp_lin.use_solver(useUmfpack = True)  # False pour superLU
```

3.2.2 Définitions des fonctions utiles pour la suite des calculs

In [ ]:
```
def d1(u,KK):
    return nd.filters.convolve(u,KK)

def d2(u,KK):
    return nd.filters.convolve(u,KK.transpose())

def grad(u,KK):
    return [d1(u,KK), d2(u,KK)]

def normalize(u):
    """
    Normalisation de l'image (les valeurs des composantes sont comprises entre 0 et 1)
    """
    u = u*1.0 - u.min()
    u = u/u.max()
    return u
def get_norme(V):
    return np.sqrt(V[0]**2 + V[1]**2)

def J_tixo(u,f,eps,g,graduf,h,KK):
    [d1u, d2u] = grad(u,KK)
    J1 = 0.5*np.sum((u-f)**2)*(h**2)
    J2 = 0.5*eps*np.sum(d1u**2 + d2u**2)*(h**2)
    J3 = g*np.sum(np.abs(graduf))*(h**2)
    return J1 + J2 + J3

def projection(norme, nsom):
    """
    La norme de sortie est t.q. :
    norme_de_sortie[i] = 1 si norme[i] < 1
    norme_de_sortie[i] = norme[i] si norme[i] > 1
    """
    test = norme > 1 # if norme > 1 => True, else => False
    index = np.compress(test,np.arange(nsom)) # on récupère les indices i t.q. norme[i] > 1
    norme_2 = np.ones(nsom)
    norme_2[index] = norme[index]
    return norme_2

def visu(vf,n,m,titre=""):
    plt.imshow(vf.reshape([n,m]), cmap=plt.cm.gray)
    plt.title(titre)

def image_arc_cercle(p, n=100, m=100):
    uns = np.array([np.ones(n)])
```

```
    x = np.array([np.linspace(0,1,n)])
    y = np.array([np.linspace(0,1,m)])
    x = uns.transpose()*x
    y = x.transpose()
    r = np.sqrt(x**2 + y**2)
    vr0 = np.linspace(0.1,0.9,p)*x.max()
    f = np.zeros([n,m])
    r0_old = 0.
    for i, r0 in enumerate(vr0):
        masque = 0.5*(np.sign(r-r0)+1)
        f += -(masque-1)*np.sin(2*np.pi*((r-r0_old)/(r0-r0_old))*2*i)
        r0_old = r0
    return f

def create_matrices(n,m, verbose=False):
    nsom = n*m
    e = np.ones(nsom)
    K = sp_sp.spdiags([e,e,-4*e,e,e],[-m,-1,0,1,m],nsom,nsom)
    M = sp_sp.eye(nsom)
    K = K.tocsc()
    if verbose:
        print "K matrice", K.shape
        print K.todense()
    return M, K

def create_matrices_with_Neumann(n,m):
    [M,K] = create_matrices(n,m)

    # Correction CL Neumann homogène :
    # Attention, cela ne fonctionne pas comme en 1D.
    # En 2D, il faut reprendre les bords gauche et droit.

    # Récupération des indices de bord
    v1 = np.arange(0,n)          # bord haut
    v2 = np.arange((m-1)*n,m*n)  # bord bas
    v3 = np.arange(0,m*n,m)      # bord gauche
    v4 = np.arange(n-1,m*n,m)    # bord droit

    # les bords sans les coins
    for i in v1[1:-1]:
        K[i,i] = -3
    for i in v2[1:-1]:
        K[i,i] = -3
    for i in v3[1:-1]:
        K[i,i-1] = 0
        K[i,i] = -3
    for i in v4[1:-1]:
        K[i,i] = -3
        K[i,i+1] = 0

    # les coins
    i = v1[0] # haut, gauche
    K[i,i] = -2
    i = v1[n-1] # haut, droit
    K[i,i] = -2
    K[i,i+1] = 0
    i = v2[0] # bas, gauche
    K[i,i] = -2
    K[i,i-1] = 0
    i = v2[n-1] # haut, droit
    K[i,i] = -2
```

```
        K = K/(h**2)
        return M, K

    def lissage_tixo(gc,eps,norme_max_grad_f,n,m,vf,h,M,K):
        """
        fonction de lissage tixotrope
        """
        nsom = n*m

        # Calcul de la matrice de lissage linéaire "A" puis factorisation de "A"
        A = M - eps*K
        A = A.tocsc()
        t0 = time.clock()
        solver = sp_sp_lin.factorized(A)
        t1 = time.clock()
        print u" factorisation effectuée en %.3f s" % (t1-t0)

        # Résolution (suite à la factorisation précédente) du problème de lissage linéaire
        t0 = time.clock()
        vu0 = solver(vf)
        t1 = time.clock()
        erreur = np.abs(vf - A.dot(vu0)).max()/np.abs(vf).max()
        print u" solution du lissage linéaire obtenue en %.3f s (après factorisation)" % (t1-t0)
        print u" avec une précision relative de %.1e" % (erreur)

        vu = vu0.copy()
        Lambda_x = np.zeros(nsom)
        Lambda_y = np.zeros(nsom)

        g = gc*eps*norme_max_grad_f
        rho = 0.1*2./(gc*norme_max_grad_f) # rho < 2*eps/g

        VJ = np.zeros(1000)

        t0 = time.clock()
        for i in xrange(100):
            graduf = grad((vu-vf).reshape([n,m]),KK)
            VJ[i] = J_tixo(vu.reshape([n,m]),f,eps,g,graduf,h,KK)
            Lambda_x = Lambda_x + rho*graduf[0].reshape(nsom)
            Lambda_y = Lambda_y + rho*graduf[1].reshape(nsom)
            norme = get_norme([Lambda_x, Lambda_y])
            norme = projection(norme, nsom)
            Lambda_x = Lambda_x/norme
            Lambda_y = Lambda_y/norme
            div_Lambda = d1(Lambda_x.reshape([n,m]),KK) + d2(Lambda_y.reshape([n,m]),KK)
            b = vf + g*div_Lambda.reshape(nsom)
            vu = solver(b)
        t1 = time.clock()

        print " Calcul tixotrope effectué en %.1f s" % (t1-t0)
        print " eps %.1e gc %.1e, rho %.1e" % (eps, gc, rho)
        return vu, vu0, VJ[0:i]
```

3.2.3 Lecture (ou création) de l'image, normalisation et création du vecteur image vf

```
In [ ]: cas = 0
        if cas == 0: f = image_arc_cercle(3, n=200, m=200) ; gc = 0.4 ; eps = 0.01
        if cas == 1: f = nd.imread("Cerveau-IRM-550.png") ; gc = 0.1 ; eps = 0.01

        f = normalize(f)
        n, m = f.shape
        if np.abs(n-m) > 0:
```

```
            sys.exit(U"on sort car le code qui suit ne gère pas le cas n différent de m !")

        vf = f.reshape(n*m)
```

3.2.4 Dimensionnement de variables utiles et visualisation

```
In [ ]: h = 1./n # taille du pixel

        # masque (pour les dérivées)
        #KK = 1./(8*h)*np.array([[-1,0,1],[-2,0,2],[-1,0,1]]) # Sobel
        KK = 1./(2*h)*np.array([[0,0,0],[-1,0,1],[0,0,0]])

        grad_f = grad(f,KK)
        norme_grad_f = get_norme(grad_f)
        # utile pour dimension le coefficient "g" associé au lissage tixotrope
        norme_max_grad_f = norme_grad_f.max()

        plt.figure(figsize=(12,5))
        plt.subplot(1,2,1)
        plt.imshow(f, cmap=plt.cm.gray)
        plt.title("f, n %d, m %d" % (n,m))
        plt.subplot(1,2,2)
        plt.imshow(norme_grad_f, cmap=plt.cm.gray)
        plt.title("norme grad f (max %.1e)" % norme_max_grad_f)
        plt.show()
```

3.2.5 Pour permettre l'utilisation de méthodes de résolution directe

- création des matrices M et K pour permettre l'utilisation de méthodes de résolution directe
- avec prise en compte de la condition limite de Neumann homogène

(avant un exemple simple pour faciliter la compréhension du fonctionnement de la fonction "create_matrices")

```
In [ ]: [M,K] = create_matrices(3,3, verbose=True)
        del M, K

In [ ]: [M,K] = create_matrices_with_Neumann(n,m)
```

3.2.6 Exécution et visualisation

```
In [ ]: [vu, vu0, VJ] = lissage_tixo(gc,eps,norme_max_grad_f,n,m,vf,h,M,K)

        plt.figure(figsize=(20,20))

        plt.subplot(2,2,1)
        visu(vf,n,m,titre="f")
        plt.subplot(2,2,2)
        visu(vu,n,m,titre=u"u tixotrope %.3f %.3f" % (vu.min(), vu.max()))

        plt.subplot(2,2,3)
        visu(vu0,n,m,titre=u"u linéaire %.3f %.3f" % (vu0.min(), vu0.max()))

        plt.subplot(2,2,4)
        plt.plot(VJ, label="J")
        plt.legend()
        plt.show()
```

3.2.7 Visualisation intéressante dans le cas 0 "image_arc_cercle"

L'image visualisée contient 4 parties :

- en haut à gauche : f,
- en haut à droite : u (l'image obtenue suite au lissage tixotrope),
- en bas à gauche : $|\nabla f|$,

- en bas à droite : $|\nabla u|$,

des rotations ainsi que des mises à l'échelle sont effectuées pour faciliter les comparaisons.

```
In [ ]: if cas == 0:
            u = vu.reshape([n,m])
            grad_u = grad(u,KK)
            norme_grad_u = get_norme(grad_u)

            image = np.zeros([2*n,2*m])

            image[0:n,0:m] = nd.rotate(f*norme_max_grad_f,-180)
            image[0:n,m:2*m] = nd.rotate(u*norme_max_grad_f,90)
            image[n:2*n,0:m] = nd.rotate(norme_grad_f,-90)
            image[n:2*n,m:2*m] = norme_grad_u

            plt.figure(figsize=(8,8))
            visu(image,2*n,2*m,titre=" f, u\n norme grad f, norme grad u")
            plt.show()
```