N° d'ORDRE : 14/2010-M/MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET

DE LARECHERCHE SCIENTIFIQUE

UNIVERSITE DES SCIENCES ET DE LA TECHNOLEGIE

HOUARI BOUMEDIENNE

FACULTE DE MATHEMATIQUES

MEMOIRE

Présenté pour l'obtention du diplôme de MAGISTER

En: MATHEMATIQUES

Spécialité : Modélisation Mathématiques et Numérique

Par : DJAFRI Samah

Sujet

Quelques problèmes inverses en traitement d'images

Soutenu publiquement le 12/12/2010, **devant le jury composé de:**

Mr- D. TENIOU,	Professeur,	à l'USTHB.	Président
Mr- T. ALIZIANE,	Maître de conférences /A,	à l'USTHB.	Dteur de Mémoire
Mr- M. BOUSSELSAL,	Professeur,	à l'ENS/ KOUBA.	Examinateur
Mr- T. Z. BOULMEZAOUD,	Maître de conférences /A,	à l'Univ. VERSAILLES.	Examinateur
Mr- M. MEDJDEN,	Maître de conférences /A,	à l'USTHB.	Examinateur

Remerciements

Je remercie très sincèrement Monsieur ***Djamel Teniou****, professeur à l'USTHB, de participer à mon jury et de me faire l'honneur d'en avoir accepté la présidence.*

Je tiens à exprimer ma profonde reconnaissance à Monsieur ***Tarik Aliziane*** *maître de conférences à l'USTHB, pour sa grande disponibilité, ses conseils et son encadrement.*

Je veux ici exprimer ma sincère gratitude à Monsieur ***Tahar Zamène Boulmezaoud*** *maître de conférences à l'université de Versailles, co-directeur de ce mémoire pour m'avoir proposé ce sujet, pour ses conseils, sa rigueur scientifique et le temps passé à débugger du code malgrés la distance.*

Je remercie vivement Monsieur ***Mahmoud Bousselsal*** *professeur à l'ENS de Kouba, ainsi que Monsieur* ***Mohamed Medjden*** *professeur à l'USTHB, d'avoir accepter d'être membre de mon jury, je les remercie également pour l'attention et le temps qu'ils ont consacrés à la lecture de ce mémoire.*

Je remercie tous mes professeurs et tous mes collègues de l'E.N.S de Kouba pour leurs aide, je cite en particulier Monsieur ***A. Ait mokhtar*** et Monsieur **A. Choutri.**

Pour finir, merci à ma famille à mes amies et à toute les personnes qui, par leur soutien moral et leurs encouragements m'ont aidé pendant ces années de recherche.

Table des matières

Introduction

Lors de l'acquisition d'une image, il arrive souvent que celle-ci soit bruitée ou partiellement endommagée. Parmi les causes possibles d'une telle dégradation, on peut citer la mauvaise réception de données, une luminosité faible ou inappropriée, une mauvaise mise au point de l'objectif, des conditions météorlogiques défavorables... Corriger l'image altérée devient dès lors souhaitable, voire nécessaire. C'est le but de la restauration d'images ; corriger au maximum les distortions causées par des effets indésirables.

D'un point de vue numérique, les dégradations que subissent les images sont la destruction ou la modification des valeurs en certains pixels (ou voxels en 3D). La réparation de ces modifications nécessite d'une part une modélisation juste de la dégradation, et d'autre part des algorithmes numériques robustes et performants. Tout cela est afin d'assurer une qualité optimale de l'image corrigée. Il existe à ce jour de nombreuses techniques, essentiellement de filtrage, qui sont employées avec plus ou moins de succès. Néanmoins, on assiste depuis le milieu des années quatre-vingt dix, à la résurgence de nouvelles méthodes basées sur l'utilisation d'équations aux dérivées partielles et qui s'avèrent de plus en plus efficaces (voir : T. F. Chan et J. Shen [15], Aubert et Kornprobst [5]).

Ce document est constitué de trois chapitres

Dans le premier chapitre, nous présentons un panorama de méthodes de restauration d'images. Le point commun de ces méthodes est de garder intact les bords des objets aperçus dans une image et d'éliminer le bruit dans les autres parties de l'image. Ces deux tâches, simples en apparence, sont pourtant extrêmement compliquées. Elles ont conduit à des modèles sophistiqués dont nous présentons certains dans ce chapitre. Les modèles dits variationnels nous intéressent tout particulièrement.

Dans le deuxième chapitre, on aborde quelques aspects théoriques concernant l'existence de solutions au modèle variationnel proposé. Les espaces $BV(\Omega)$ seront utilisés comme principal cadre fonctionnel.

Le troisième chapitre est consacré aux aspects pratiques, qui constituent le coeur de ce travail. Après avoir présenté ces méthodes, pour les problèmes de débruitage (denoising) et de défloutage (deblurring), on expose quelques résultats obtenus sur des images réelles avec nos codes écrits en Langage C ou sous matlab.

Chapitre 1

Modélisation du problème

Dans tout ce qui suit, une image sera modélisée par la donnée d'une intensité réelle u définie comme une fonction (scalaire par simplicité) sur un domaine Ω de $\mathbb{R}^n$ (n est la dimension de l'image).

La restauration est l'un des objectifs majeurs du traitement des images. Une image dégradée est une image qui a subi une altération affectant sa qualité. Les questions de restauration d'images auxquelles on s'intéresse ici sont celles qui consistent à améliorer une image observée issue d'une image dégradée par l'effet d'un bruit ou d'un flou. On parle alors de débruitage (denoising) et de défloutage (deblurring) ou deconvolution.

1.1 Le bruit et le flou

1.1.1 Le bruit

Le bruit caractérise les parasites ou interférences d'un signal, c'est-à-dire les parties du signal déformées localement. Ses causes peuvent être nombreuses. On peut citer le contexte d'acquisition (sur ou sous illumination, qualité ou perturbations des capteurs,...etc), les bruits de transmission,..etc. Il existe bien entendu divers types de bruits qui peuvent être classifiés indépendamment de leurs causes. Le bruit dit "poivre et sel" par exemple consiste en l'ajout de pixels noirs (ou blancs) aléatoirement dans une image. Le bruit Gaussien est obtenu en ajoutant à chaque pixel une valeur aléatoire suivant une loi de probabilité Gaussienne :

$$\rho(s) = \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-|s-\mu|^2}{2\sigma^2}. \tag{1.1}$$

D'un point de vue proprement mathématique, on se restreint ici aux bruits additifs, c'est-à-dire qui viennent s'ajouter à l'image originale. Ainsi, si on note u l'image original et u_0 l'image modifiée (ou bruitée), on peut écrire

$$u_0 = u + n,$$

où n est le bruit, considéré comme une variable aléatoire.

Dans les figures (1.2) et (1.3), on donne l'exemple de deux images bruitées à partir de l'image originale illustrée dans la figure (1.1). La première comporte un bruit de type poivre et sel, tandis que la seconde comporte un bruit gaussien.

FIGURE 1.1 – Image de Barbara originale non bruitée

FIGURE 1.2 – Image de Barbara modifiée avec un bruit de type poivre et sel avec une densité de 0.05

FIGURE 1.3 – Image de Barbara modifiée avec un bruit gaussien de variance $\sigma = 20$ et de moyenne $\mu = 0.008$

1.1.2 Le flou

La deuxième cause de dégradation d'images, à laquelle on s'intéresse ici, est le flou. Le flou dans une image peut être dû à plusieurs causes ; il peut être en effet la conséquence du phénomène de défocalisation propre à l'appareil photo, ou à un mouvement de l'utilisateur lors de la prise de l'image ou encore aux conditions météorologiques. En pratique, on peut considérer que la dégradation que subit une image donnée u se traduit par l'action d'un opérateur R, souvent considéré comme linéaire convolutif

$$Ru = k \star u,$$

où k est un noyau de convolution appelé souvent le PSF (Point Spread Function) ; il peut-être connu ou inconnu. Dans le cas d'un flou isotrope de défocalisation (out-of-focus), il s'écrit sous la forme

$$k(x) = \frac{1}{V_n} 1_{B_r}(x),$$

où B_r est la boule de rayon r, et V_n son volume.

Quand ce flou est Gaussien, ce PSF s'écrit

$$k(x) = \frac{1}{(2\pi)^{n/2}\sigma^n} \exp^{-|x|^2/(2\sigma^2)}.$$

Ainsi, l'action superposée d'un bruit additif n et d'un opérateur de flou R conduit au modèle

$$u_0 = Ru + n,$$

où u est l'image originale et u_0 l'image observée.

1.1.3 Mesure du bruit et de la qualité de restauration : le SNR et l'INSR

Avant d'exposer les modèles utilisés pour débruiter ou déflouter une image, soulignons une manière habituelle d'estimer l'impact du bruit n dans une image dégradée u_0 est d'évaluer le SNR (Signal to Noise Ratio), rapport des variances de l'image dégradée et du bruit (exprimée en décibels)

$$SNR = 10 \log_{10} \frac{Var(u_0)}{Var(n)}.$$

Si ce SNR est supérieur à 20, l'image est considérée comme de bonne qualité. S'il est inférieur à 10, elle est considérée comme étant de mauvaise qualité.

Notons aussi qu'afin de quantifier la qualité de la restauration pour une image bruitée, on utilise parfois une mesure appelée l'ISNR (Improvement Signal to Noise Ratio). Soit u l'image originale, u_0 l'image modifiée et f l'image restaurée. Alors l'ISNR est définie par :

$$ISNR(u, f, u_0) = 10 \log_{10} \frac{\sum_{i,,j}(u(i,j) - u_0(i,j))^2}{\sum\limits_{i=-\infty}^{+\infty} \sum_{i,j}(u(i,j) - f(i,j))^2} \tag{1.2}$$

Si l'image corrigée est proche de l'image originale, alors le dénominateur va être proche de "0", et donc l'ISNR sera proche de l'infini.
Si l'image corrigée est exactement identique à l'image bruitée, alors la fonction sera égale à 1, donc l'ISNR sera égal à "0".
Enfin, si l'image restaurée est très différente de l'image originale, le dénominateur sera assez important, l'ISNR tendra vers moins l'infini. On peut résumer :

- Si l'ISNR est négatif, la restauration est mauvaise.
- Si l'ISNR est nul, alors il n'y a pas eu de restauration (on reste proche de l'image dégradée).
- Si l'ISNR est positif, alors la restauration est de bonne qualité.

1.2 Modèles de restauration

Nous allons exposer ici quelques modèles de restauration de type variationnels ; ce sont ces modèles qui nous intéressent ici. Néanmoins, nous consacrerons un petit paragraphe en fin de ce chapitre pour énumérer quelques méthodes EDP.

1.2.1 Les modèles variationnels

En l'absence du flou, une façon déterministe d'éliminer un bruit n, supposé en général gaussien, s'appuie sur le principe de Maximum de vraissemblance et consiste habituellement à résoudre le problème

$$\min_u \int_\Omega |u - u_0|^2 dx + \lambda \int_\Omega \phi(|\nabla u|) dx, \tag{1.3}$$

où le premier terme mesure la fidélité par rapport à l'image observée ; tandis que le second terme est un terme de régularisation. Le choix de la fonction ϕ n'est pas quelconque et sera discuté ultérieurement.

En présence d'un flou dû à un opérateur R connu a priori, cette méthode variationnelle pour reconstruire une image proche de l'image originale consiste à minimiser la fonctionnelle

$$\min_u \int_\Omega |Ru - u_0|^2 dx + \lambda \int_\Omega \phi(|\nabla u|) dx. \tag{1.4}$$

On parle alors de déconvolution (ou de deblurring).
Si par contre l'opérateur de flou R n'est pas connu entièrement, et dépend d'un paramètre p, inconnu en pratique, on peut chercher le couple (u, p) en résolvant le problème de minimisation

$$\min_{(u,p)} \int_\Omega |R(p)u - u_0|^2 dx + \lambda \int_\Omega \phi(|\nabla u|) dx. \tag{1.5}$$

On parle alors de *déconvolution aveugle*, car seule l'image observée est connue, le noyau de convolution étant en géneral inconnu. Le problème est de calculer une approximation de l'image réelle discrète, ne connaissant que la structure de la dégradation. Il nous faut donc calculer en même temps l'image initiale et les paramètres du bruit ou du flou.

Le problème (1.5) est souvent mal posé et doit alors être régularisé par rapport à p. Il est aussi souvent non convexe par rapport au couple (u, p) (mais convexe néanmoins par rapport à u et même par rapport à p).

Dans ce mémoire nous nous placerons essentiellement dans le cas où l'opérateur du flou est connu. Une importance particulière sera accordée au problème du débruitage ($R = I$). Les modèles de type (1.3) ou (1.4) seront bien détaillés dans la suite.

Nous allons discuter maintenant brièvement le choix de la fonction ϕ. Pour cela, gardons à l'esprit les deux objectifs qualitatifs suivants de la restauration

– on voudrait d'une part préserver les contours de l'image, en éliminant les effets d'un éventuel lissage de ces contours,
– on aimerait lisser les zones d'intensité homogène.

D'un point de vue mathématique, on pourrait considérer que les contours sont les régions de fortes variations ou de discontinuités de l'intensité u de l'image, c'est-à-dire celle pour laquelle ∇u est de norme élevée. Par contre, les zones d'intensité homogène correspondent, quant à elles, aux régions où ∇u est de norme faible.

Revenons au problème (1.4). Il s'écrit sous la forme

$$\min_u F(u) = \int_\Omega L(x, u(x), \nabla u(x))dx \tag{1.6}$$

où $L \;:\; (x, v, w) \mapsto L(x, v, w)$ est une fonction de $\Omega \times \mathbb{R} \times \mathbb{R}^n$ dans $\mathbb{R}$. On a *formellement,*

$$\begin{aligned}
F(u+h)-F(u) &= \int_\Omega [L(x,u(x)+h(x),\nabla u(x)+\nabla h(x))-L(x,u(x),\nabla u(x))]dx \\
&= \int_\Omega (\frac{\partial L}{\partial v}(x,u(x),\nabla u(x))h(x)+\nabla_w L(x,u(x),\nabla u(x)).\nabla h(x))dx \\
&\quad +o(\|h\|^2) \\
&= \int_\Omega [\frac{\partial L}{\partial v}(x,u(x),\nabla u(x))-\mathrm{div}_x\nabla_w L(x,u(x),\nabla u(x))].h(x)dx \\
&\quad +\int_{\partial\Omega}\nabla_w L(x,u(x),\nabla u(x)).nh(x)dx+o(\|h\|^2)
\end{aligned}$$

et on a

$$\begin{aligned}
dF(u).h &= \int_\Omega [\frac{\partial L}{\partial v}(x,u(x),\nabla u(x)) \\
&\quad -\mathrm{div}_x\nabla_w L(x,u(x),\nabla u(x))]h(x)dx \\
&\quad +\int_{\partial\Omega}(\nabla_w L(x,u(x),\nabla u(x)).n)h(x)dx.
\end{aligned}$$

La condition d'optimalité s'écrit $dF(u)=0$ se traduit formellement par l'équation

$$\frac{\partial L}{\partial v}(x,u(x),\nabla u(x))-\mathrm{div}\nabla_w L(x,u(x),\nabla u(x))=0 \tag{1.7}$$

complétée avec la condition aux limites

$$\nabla_w L(x,u(x),\nabla u(x)).n=0 \text{ sur } \Omega.$$

L'équation (1.7) est appelée l'équation d'Euler-Lagrange associée au problème variationnel (1.6).

En particulier, l'équation d'Euler-Lagrange associée au problème variationnel (1.4), s'écrit de la manière formelle suivante

$$R^*Ru-\lambda\mathrm{div}\left(\frac{\phi'(|\nabla u|)}{|\nabla u|}.\nabla u\right)=R^*u_0 \tag{1.8}$$

avec les conditions aux bords de Ω

$$\frac{\phi'(|\nabla u|)}{|\nabla u|}.\frac{\partial u}{\partial n}=0 \tag{1.9}$$

En décomposant l'opérateur de divergence en tout point x tel que $|\nabla u| \neq 0$, on obtient ce qui suit ;

$$\text{div}\left(\frac{\phi^{'}(|\nabla u|)}{|\nabla u|}.\nabla u\right) = \frac{\partial}{\partial x}\left(\frac{\phi^{'}(|\nabla u|)}{|\nabla u|}\right).u_x + \frac{\partial}{\partial y}\left(\frac{\phi^{'}(|\nabla u|)}{|\nabla u|}\right).u_y + \frac{\phi^{'}(|\nabla u|)}{|\nabla u|}.\Delta u.$$

Or, on sait que :

$$\frac{\partial}{\partial x}\phi^{'}(|\nabla u(x)|) = \phi^{''}(|\nabla u|)(u_x u_{xx} + u_y u_{xy}).$$

on en déduit donc que :

$$\begin{aligned}
\frac{\partial}{\partial x}\left(\frac{\phi^{'}(|\nabla u|)}{|\nabla u|}\right) &= \frac{1}{|\nabla u|^2}\left(\frac{\partial \phi^{'}(|\nabla u|)}{\partial x}.|\nabla u|\right) - \left(\frac{\phi^{'}(|\nabla u|)}{|\nabla u|}\right)(u_x u_{xx} + u_y u_{xy}) \\
&= \frac{1}{|\nabla u|^2}\left(\phi^{''}(|\nabla u|) - \frac{\partial \phi^{'}(|\nabla u|)}{|\nabla u|}\right)(u_x u_{xx} + u_y u_{xy})
\end{aligned}$$

ainsi :

$$\begin{aligned}
\text{div}\left(\frac{\phi^{'}(|\nabla u|)}{|\nabla u|}.\nabla u\right) &= \frac{1}{|\nabla u|^2}\left(\phi^{''}(|\nabla u|) - \frac{\partial \phi^{'}(|\nabla u|)}{|\nabla u|}\right)(u_x^2 u_{xx} + 2u_x u_y u_{xy} + u_y^2 u_{yy}) \\
&+ \frac{\phi^{'}(|\nabla u|)}{|\nabla u|}.\triangle u.
\end{aligned}$$

Sachant que les dérivées secondes directionnelles u_{NN} et u_{TT} sont données par

$$\frac{\partial^2 u}{\partial N^2} = \frac{1}{|\nabla u|^2}(u_x^2 u_{xx} + 2u_x u_y u_{xy} + u_y^2 u_{yy})$$

et

$$\frac{\partial^2 u}{\partial T^2} = \frac{1}{|\nabla u|^2}(u_x^2 u_{yy} - 2u_x u_y u_{xy} + u_y^2 u_{xx})$$

On a aussi : $\Delta u = \dfrac{\partial^2 u}{\partial N^2} + \dfrac{\partial^2 u}{\partial T^2}$, d'où

$$div\left(\frac{\phi^{'}(|\nabla u|)}{|\nabla u|}.\nabla u\right) = \phi^{''}(|\nabla u|)\frac{\partial^2 u}{\partial N^2} + \frac{\phi^{'}(|\nabla u|)}{|\nabla u|}.\frac{\partial^2 u}{\partial T^2}. \tag{1.10}$$

On obtient formellement en $2D$:

$$R^*(Ru - u_0) - \lambda\left\{\frac{\phi^{'}(|\nabla u|)}{|\nabla u|}u_{TT} + \phi^{''}(|\nabla u|)u_{NN}\right\} = 0 \tag{1.11}$$

où u_{NN} est la dérivée, seconde dans la direction du gradient $N = \frac{\nabla u}{|\nabla u|}$ et u_{TT} la dérivée seconde dans la direction T orthogonale au gradient N, tangente à la courbe de niveau u en x.

L'écriture (1.11) permet clairement de distinguer les deux termes de diffusion ; celle tangentielle aux contours et celle qui est transverse.

Finalement, les conditions portent sur le comportement de la fonction ϕ se traduisent par :

– (i) On aimerait lisser par diffusion que dans les zones de faible gradient ($|\nabla u|$ proche de 0), c'est-à-dire

$$\lim_{t \to 0} \frac{\phi^{'}(t)}{t} = m > 0, \text{ et } \phi^{'}(0) = 0.$$

– (ii) On aimerait éviter la diffusion transverse dans les contours, c'est-à-dire les zones de forts gradients. D'où

$$\lim_{t \to \infty} \frac{\phi^{'}(t)}{t} = M > 0, \text{ et} \lim_{t \to +\infty} \phi^{''}(t) = 0.$$

Ces deux conditions sont complétées parfois par une condition mathématique :

– (iii) $\dfrac{\phi^{'}(t)}{t}$ continue et strictement décroissante afin d'assurer la stabilité du terme de régularisation.

Notons les deux cas

1. Régularisation de Tykhonov : Elle correspond au choix $\phi(t) = t^2$. Elle fût l'une des premières régularisations utilisées. Elle ne vérifie pas la condition (ii), c'est-à-dire qu'elle ne préserve pas les discontinuités de la solution. De plus, si l'on regarde les valeurs que prennent les coordonnées du terme de régularisation d'après l'équation (1.11) :

$$-\text{div}\left(\frac{\phi^{'}(|\nabla u|)}{|\nabla u|}.\nabla u\right) = -2\frac{\partial^2 u}{\partial N^2} - 2\frac{\partial^2 u}{\partial T^2}. \tag{1.12}$$

Et, l'équation (1.11) s'écrit de nouveau de la manière suivante

$$R^*Ru - R^*u_0 = 2\lambda\frac{\partial^2 u}{\partial N^2} + 2\lambda\frac{\partial^2 u}{\partial T^2}. \tag{1.13}$$

Le lissage est donc bien effectué de manière isotropique sur tous les points de la solution.

En conclusion, avec cette régularisation les contours sont peu à peu estompés et l'image reconstruite est floue. Cela explique pourquoi cette régularisation élimine les contours et donne des images d'une qualité non satisfaisante.

2. Régularisation TV ou ROF (modèle de Rudin -Osher-Fatemi [39]) : Elle correspond au choix $\phi(t) = t$. C'est l'une des régularisations les plus étudiées et les plus utilisées en littérature. La régularisation par Variation Totale, ne vérifie pas la condition (i) présentée avant.

   Le lissage qu'elle effectue n'est donc pas isotropique lorsqu'on se situe d'une zone homogène. En effet, la dérivée du terme de régularisation présentée dans (1.10), devient :

$$-\text{div}\left(\frac{\phi'(x)(|\nabla u(x)|)}{|\nabla u(x)|}.\nabla u(x)\right) = -\text{div}\left(\frac{\nabla u(x)}{|\nabla u(x)|}\right) = -\frac{u_{TT}(x)}{|\nabla u(x)|} \tag{1.14}$$

   où u_{TT} est la dérivée seconde selon la direction T tangentielle à la courbe de niveau au point x.

   Et, l'équation (1.11) s'écrit dans ce cas de la manière suivante

$$R^*Ru - R^*u_0 = \lambda\frac{u_{TT}(x)}{|\nabla u(x)|}. \tag{1.15}$$

   Le lissage ne peut donc se faire en chaque point de l'image que le long de la courbe de niveau à laquelle il appartient. Cette propriété est utile lorsqu'on se situe sur un des contours de l'image : aucun lissage n'est effectué perpendiculairement à ce contour. Mais lorsqu'on se situe dans une zone homogène, le lissage est toujours effectué selon la direction perpendiculaire à la courbe de niveau. On privilégie donc, dans ce cas, une direction particulière alors que, raisonnablement le lissage d'une zone homogène se voudrait être plutôt isotropique.

La régularisation par Variation Totale fera l'objet d'une étude détaillée dans ce mémoire.

D'autres types de régularisation sont exposés dans le tableau (1.1).

Nom	convexité	$\phi(u)$	$\phi^{'}(u)/2u$	Condition
ϕ_Q Quadratique	oui	u^2	1	non
$\phi_{\mathbb{L}^1}$ Norme $\mathbb{L}^1$	oui	$\|u\|$	$\frac{1}{\|u\|}$	non
ϕ_{BS} Bouman, Sauer	oui	$t^\alpha, 1 \leq \alpha \leq 2$	$\alpha u^{\alpha-2}/2$	non
ϕ_{PM} Perona, Malik	non	$1 - \exp(-t^2)$	$exp(-u^2)$	oui
ϕ_{QT} Quadratique tronquée	non	$\min(u^2, 1)$	0 si $u < 1$,0 si$u \geq 1$	oui
ϕ_{GM} German, McClure	non	$u^2/1 + u^2$	$\frac{1}{(1+u^2)^2}$	oui
ϕ_{HL} Hebert, Leahy	non	$\log(1 + u^2)$	$\frac{1}{(1+u^2)}$	oui
ϕ_{GR} Green	oui	$\log(\cosh(u))$	$\frac{\tanh(u)}{u}$	oui
ϕ_{HS} Hyper Surfaces	oui	$2\sqrt{1 + u^2} - 2$	$\frac{1}{\sqrt{1+u^2}}$	oui

TABLE 1.1 – Fonctions de régularisation ϕ avec leurs fonctions $\frac{\phi^{'}(t)}{t}$ associées.

Précisons enfin que le choix du paramètre λ permet de fixer l'importance que l'on va accorder au terme de régularisation par rapport au terme d'attache aux données. En choisissant une forte valeur pour ce paramètre, on privilégie l'a priori apporté par la régularisation par rapport au terme d'attache aux données. Ce choix devient nécessaire dans le cas où les données sont fortement bruitées.

L'inconvénient est qu'on risque de reconstruire un objet flou, assez éloigné de l'objet réel, si l'a priori utilisé n'est pas assez précis. En choisissant une faible valeur pour ce paramètre, on privilégie le terme d'attaches aux données par rapport au terme de régularisation. L'inconvénient est qu'on risque de reconstruire un objet bruité. Autrement dit, le choix de ce paramètre λ est fait souvent de façon empirique. Dans la suite de ce mémoire, nous supposerons, d'un point de vue proprement mathématique, que λ est fixé.

1.2.2 Les méthodes EDP

Les méthodes variationnelles exposées dans le paragraphe précédent peuvent en quelque sorte être considérées comme des méthode d'EDP au sens où elles conduisent à la résolution de l'équation d'Euler-Lagrange (1.8) qui est une équation aux dérivées partielles. Il existe néanmoins d'autres méthodes basées sur des équations aux dérivées partielles écrites directement.

Une grande famille parmi ces méthodes consiste à faire évoluer dans le temps une fonction à support continu u, selon l'équation de la forme

$$\begin{cases} \dfrac{\partial u}{\partial t} = F(x, u(t,x), \nabla u(t,x), \nabla^2 u(t,x))), & (o,T) \times \Omega \\ \dfrac{\partial u}{\partial N} = 0, & \text{sur } \partial\Omega \\ u(x,0) = u_0(x). \end{cases} \tag{1.16}$$

Le choix des paramètres de cette équation est fait de sorte à éviter la diffusion transverse aux contours (dans le sens du gradient) pour ne pas le détériorer ; au voisinage des contours cette diffusion est essentiellement parallèle aux contours. D'autre part, le bruit est lissé par diffusion le long des contours.

Par exemple, l'équation de la chaleur fût l'une des équations utilisées car elle permet d'effectuer une diffusion isotrope. Elle a pour effet de lisser l'image

(donc le bruit)

$$\begin{cases} \dfrac{\partial u}{\partial t} = \Delta u, & t \geq 0,\ x \in \Omega \\ \dfrac{\partial u}{\partial N} = 0, & \text{sur } \partial\Omega \\ u(x,0) = u_0(x). \end{cases} \tag{1.17}$$

L'inconvénient de ce modèle est que cette EDP introduit une diffusion isotrope qui s'opère de manière identique dans toutes les directions et ne possède aucune direction privilégiée. Ainsi, dans des régions d'intensité homogène, ce processus permettra de réduire effectivement l'effet du bruit. Mais, dans des régions présentant des discontinuités au niveau de l'intensité, celles-ci seront aussi lissées et le contraste visuel de ces parties sera sensiblement réduit.

Donc, pour préserver les contours tout en lissant le bruit, il est préférable d'effectuer une diffusion anisotrope. C'est ainsi que Malik et Perona (voir [34]) proposent de remplacer l'équation de la chaleur par l'équation aux dérivées partielles non linéaire, suivante :

$$\frac{\partial u}{\partial t}(x,t) = \operatorname{div}(g(|\nabla u(t,x)|)\nabla u(t,x)) \tag{1.18}$$

avec les conditions aux bords

$$\begin{cases} \dfrac{\partial u}{\partial N} = 0, \ \text{sur } \partial\Omega \\ u(x,0) = u_0(x), \end{cases} \tag{1.19}$$

où g est une fonction régulière à valeurs positives, qui vérifie

$$g(0) = 1, \lim_{s\to+\infty} sg(s) = 0 \text{ et } g^{'}(s) \leq 0 \text{ pour tout } s > 0. \tag{1.20}$$

Le coefficient de conduction $g(.)$ est égal à 1 dans les zones homogènes (à faible gradient) et tend vers zéro dans les zones de contours et la diffusion est ainsi stoppée au niveau des contours. L'image finale est une image obtenue au terme d'un temps t_u à determiner. Ce temps est usuellement fixé par l'utilisateur.

Malheureusement, ce modèle présente une difficulté majeure qui réside dans son cadre théorique, puisque toute fonction g vérifiant les conditions (1.20), rend le problème mal posé. Une solution pour résoudre ce problème d'instabilité, est de travailler avec une version régularisée de l'équation impliquant le gradient $g(|\nabla G_\sigma * u|)$ au lieu de $g(|\nabla u|)$. Cette solution rend le problème bien posé et les résultats stables. Toutefois, on peut noter que cette nouvelle EDP possède aussi quelques inconvénients, par exemple la stabilité de ce modèle n'est généralement pas garantie quand le paramètre t tend vers 0.
Une amélioration de cette idée de diffusion anisotrope qui prend en compte ces remarques a été ainsi proposée par Alvarez, Guichard, Lions et Morel [2] qui ont étudié une classe d'EDP paraboliques non linéaires qui généralisent l'idée de diffusion anisotrope proposée par Perona et Malik.
Le schéma proposé est issu de l'équation suivante :

$$\begin{cases} \dfrac{\partial u}{\partial t} = g(|\nabla G_\sigma * u|)|\nabla u|\mathrm{div}(\dfrac{\nabla u}{|\nabla u|}) \\ u(x,0) = u_0(x), \end{cases} \tag{1.21}$$

où $u_0(x)$ constitue l'image initiale à traiter, $u(x,t)$ est l'image lissée à l'échelle référencée par le paramètre t et $g(|\nabla u|)$ une fonction non croissante de la variable $|\nabla u|$ qui tend vers 0 quand la variable $|\nabla u|$ tend vers l'infini. En remarquant que le terme $|\nabla u|\mathrm{div}(\frac{\nabla u}{|\nabla u|})$ correspond à la dérivée second de u dans la direction orthogonale au gradient $|\nabla u|$. L'équation (1.21) peut s'interpréter comme un lissage anisotrope conditionnel mais seulement le long des courbes de niveaux de l'image $u(x,t)$. Le terme impliquant la fonction $g(|.|)$ permet de contrôler la vitesse de diffusion dans les zones où le gradient est faible ; ce terme est grand et permet une diffusion anisotrope le long de la direction orthogonale au gradient. Dans les zones où le gradient est fort, la pondération est faible et annule la diffusion ; d'où le nom de lissage sélectif donné par les auteurs à ce schéma.

Nodström [36] propose de minimiser la fonctionnelle suivante :

$$E(u,v) = \int_\Omega (u-u_0)^2 + \int_\Omega v|Du|^2 + \int_\Omega \lambda^2(v - \ln v) \tag{1.22}$$

$v : \Omega \to [0,1]$ est une representation des bords de l'image : proche de 1 dans les zones homogènes et de 0 au voisinage des bords. Le premier terme garantit une certaine conformité de u à l'image initiale u_0, et le deuxième régularise u dans les zones homogènes (quand v est proche de 1). Le dernier terme sert à mesurer la variable du niveau de gris de l'image. Les équations d'Euler correspondantes s'écrivent :

$$\begin{cases} (u-u_0)^2 - \operatorname{div}(v\nabla u) = 0 \\ \lambda^2\left(1-\frac{1}{v}\right) + |\nabla u|^2 = 0, \end{cases} \tag{1.23}$$

L'equation précédente nous permet d'écrire v sous la forme suivante :

$$v = g(u) = \frac{1}{1+|\nabla u|^2}.$$

Les équations (1.23) s'interprètent comme étant l'état stationnaire du modèle de réaction-diffusion suivant :

$$\begin{cases} \frac{\partial u}{\partial t} = div\left(\dfrac{\nabla u}{1+(\frac{|\nabla u|}{\lambda})^2}\right) - (u-u_0) \\ u(.,0) = u_0, \end{cases} \tag{1.24}$$

En fait, cette équation d'évolution représente un compromis entre la diffusion anisotropique proposée par Malik et Perona et le terme d'attraction vers la donnée initiale $(u-u_0)$. L'avantage de ce formalisme est que même si le temps d'arrêt est trop grand, l'image résultat reste relativement proche de l'image initiale. L'instabilité de ce modèle est évidente puisque la fonctionnelle g n'est pas convexe, elle peut posséder plusieurs minima locaux et le processus peut ne pas converger vers un minimum global.

1.3 Vers la déconvolution aveugle

Bien que cela dépasse les objectifs de ce mémoire, souligons qu'il existe des modèles plus généraux dans lesquels le noyau du flou lui même est inconnu. Ces modèles ne sont pas encore bien murs dans la littérature et restent encore sujets à de nombreuses recherches.

En résumé, on peut observer que dans les modèles variationnels exposés auparavant et qui seront étudiés dans les deux prochains chapitres, on a supposé que l'origine du flou, c'est-à-dire l'opérateur R, est connu à l'avance. En pratique, c'est rarement le cas ; il est en conséquence nécessaire de développer aussi des techniques pour estimer l'action de cet opérateur R et éliminer en conséquence le flou qui en résulte. On rappelle que souvent R est un opérateur de convolution de la forme

$$Ru = k \star u,$$

où le noyau k est souvent inconnu. Dans beaucoup de cas, il est souhaitable de reconstruire k au même temps que l'image originale ; il s'agit d'un problème de déconvolution aveugle, c'est-à-dire sans connaissance du noyau de convolution. Un des modèles consiste à minimiser la fonctionnelle double régularisée suivante :

$$F(u,k) = \|u * k\|_Y^2 + \alpha_1 \|u - \overline{u}\|_{X_1}^{\gamma_1} + \alpha_2 \|k - \overline{k}\|_{X_2}^{\gamma_2}, \tag{1.25}$$

où X_1, X_2 et Y sont des espaces de Banach, $\overline{u}$ et $\overline{k}$ sont des estimations à priori de l'image et du bruit respectivement, γ_1, γ_2 sont des constantes positives et

α_1, α_2 deux paramètres positifs de régularisation.
On peut montrer que sous certaines hypothèses, ce problème admet une solution (voir [33]).

Le modèle (1.25) nécessite en pratique une connaissance d'une estimation à priori de l'image et du bruit. Mais cette information n'est pas toujours accessible. Un autre modèle (voir [15]) consiste tout simplement à minimiser la fonctionnelle suivante :

$$E(u,k) = \frac{1}{2}\int_{\mathbb{R}^2} (k * u - u_0)^2 dx + \alpha_1 \int_{\Omega} |Du| dx + \alpha_2 \int_{\mathbb{R}^2} |Dk| dx. \tag{1.26}$$

Les équations d'Euler-Lagrange associées à ce modèle s'écrivent formellement

$$\begin{aligned} u(-x,-y) * (u * k - z) - \alpha_2 \nabla.\left(\tfrac{\nabla k}{|\nabla k|}\right) &= 0, \\ k(-x,-y) * (k * u - z) - \alpha_1 \nabla.\left(\tfrac{\nabla u}{|\nabla u|}\right) &= 0. \end{aligned}$$

Cette approche a été proposée dans [15], sans preuve solide de sa performance en pratique.

Nous n'aborderons pas par la suite les problèmes de déconvolution aveugle.

Chapitre 2

Le modèle variationnel continu

Le but de ce chapitre est d'esquisser l'étude du problème de restauration d'un point de vue continu, c'est-à-dire en considérant l'intensité de l'image u comme une fonction (ou une distribution) définie sur le domaine de l'image $\Omega \subset \mathbb{R}^2$. Le cas discret selon lequel l'intensité u est un tableau fini de valeurs associées à des pixels sera discuté dans le chapitre suivant.

Par ailleurs, on se placera ici dans le cas où l'opérateur de floutage (ou de convolution) R est connu.

Considérons dans un premier temps le problème de minimisation

$$\min_u \int_\Omega |Ru - u_0|^2 + \lambda \int_\Omega \phi(|\nabla u|)dx, \tag{2.1}$$

Pour étudier ce problème, on pourrait penser à utiliser l'espace

$$W = \{u \in L^2(\Omega),\ \nabla u \in L^1(\Omega)^2\},$$

Malheureusement, cet espace n'est pas reflexif et on ne dispose pas de résultat concernant la convergence de suites bornées dans cet espace (et en particulier les suites minimisantes). De même, les espaces de Sobolev usuels $W^{1,p}(\Omega)$ ne sont pas non plus adéquats pour étudier ces problèmes car ils ne permettent pas de gérer les discontinuités facilement. En effet, près des discontinuités d'une image le gradient de l'intensité ∇u n'est pas considéré comme une fonction mais comme une mesure. Il est donc inapproprié de chercher u dans $W^{1,p}(\Omega)$ car dans ce cas ∇u est une fonction.

Les espaces considérés comme adéquats pour étudier les problèmes de type (2.1) sont les espaces des fonctions à variations bornées ($BV(\Omega)$) car ils pallient à ces difficultés. Néanmoins, il est nécessaire de remodeler dans ce cas le problème considéré (2.1) pour qu'il ait un sens, et d'autre part pour que la fonctionnelle considérée soit semi-continue inférieurement.

Nous allons dans un premier temps introduire de façon résumée les espaces BV en énumérant quelques unes de leurs propriétés.

2.1 Les espaces BV

Dans ce qui suit Ω est un ouvert borné de $\mathbb{R}^N$ de frontière Lipschitz et $\mathcal{C}_c^1(\Omega, \mathbb{R}^2)$ est l'espace des fonctions $\mathcal{C}^1$ à support compact dans Ω à valeur dans $\mathbb{R}^2$.

Définition 2.1 *Une fonction $f \in L^1(\Omega)$ à valeur dans $\mathbb{R}$ est dite à variation bornée si*

$$\sup \left\{ \int_\Omega u(x) \mathrm{div} \phi(x), \;\; \varphi \in \mathcal{C}_c^1(\Omega, \mathbb{R}^2); \; \|\varphi\|_\infty \leq 1 \right\} < +\infty, \tag{2.2}$$

On note $BV(\Omega)$ l'espace des fonctions de $L^1(\Omega)$ à variations bornées.

L'espace $BV(\Omega)$ est doté de la norme

$$\|u\|_{BV(\Omega)} = \|u\|_{L^1} + \int_\Omega |Du|,$$

où, par définition,

$$\int_\Omega |Du| = \sup\left\{ \int_\Omega u(x)\mathrm{div}\phi(x),\ \ \varphi \in \mathcal{C}_c^1(\Omega, \mathbb{R}^2);\ \|\varphi\|_\infty \leq 1 \right\}.$$

est un espace de Banach.

Notre objectif ici n'est pas de redémontrer toutes les propriétés de ces espaces. Nous nous contentons de celles qui sont utiles pour l'étude de notre problème. Pour plus de détails, le lecteur peut consulter aux réferences ([3], [22], [28]).

On a le résultat suivant important pour la caractérisation

Lemme 2.1.1 *Soit $u \in BV(\Omega)$. Alors, il existe une mesure de Radon positive μ sur Ω et une fonction $\mu-$mesurable ; $\sigma : \Omega \to \mathbb{R}$ telle que.*

i) $|\sigma(x)| = 1\ \mu\, p.p$

ii) $\int_\Omega u(x)\mathrm{div}\varphi(x)dx = -\int_\Omega \varphi(x)\sigma(x)d\mu$ *pour toute fonction* $\varphi \in \mathcal{C}_c^1(\Omega, \mathbb{R}^2)$.

Ainsi, pour toute fonction $u \in BV(\Omega)$, Du, le gradient de u au sens des distributions peut être considéré comme une mesure de Radon à valeur vectorielle $Du = \sigma d\mu$.

Ce lemme permet d'introduire la topologie faible $\star$ dans l'espace $BV(\Omega)$ de la manière suivante : une suite (u_k) converge $BV - w\star$ si

- u_k converge fortement vers u dans $L^1(\Omega)$
- $\int_\Omega \varphi Du_k$ converge vers $\int_\Omega \varphi Du$ pour tout $\varphi \in C_0(\Omega)^n$.

Cette topologie a de meilleures propriétés de compacité que la topologie forte dans BV. On a en particulier la propriété suivante :

Lemme 2.1.2 *De toute suite $(u_n)_n$ uniformément bornée dans $BV(\Omega)$ on peut extraire une sous suite qui converge $BV - w\star$ dans $BV(\Omega)$.*

Voici un ensemble de propriétés importantes de ces espaces

- Pout tout réel $p \geq 1$ tel que $p \leq N/(N-1)$ si $N > 1$, on a l'injection $BV(\Omega) \hookrightarrow L^p(\Omega)$. Cette injection est compacte quand $N = 1$ ou quand $N > 1$ et $p < N/(N-1)$.
- L'injection $BV(\Omega) \hookrightarrow L^1(\Omega)$ est compacte.
- Soit $(u_n)_n$ dans $BV(\Omega)$ tel que $u_n \to u$ dans $L^1(\Omega)$. Alors,

$$\int_\Omega |Du| \leq \liminf_{n \to +\infty} \int_\Omega |Du_n|.$$

  Il s'agit là d'une prorpriété de semi-continuité inférieure.
- si on suppose que Ω est connexe, alors toute fonction $u \in BV(\Omega)$ vérifiant $Du = 0$ est constante.

Théorème 2.1 *Pour tout $u \in BV(\Omega)$, on peut trouver une suite (u_n) telle que :*

1. $u_n \in C^\infty(\Omega)$ *pour* $n = 1, 2...$
2. $u_n \to u$ *dans* $L^1(\Omega)$.
3. $\int_\Omega |Du_n| \to \int_\Omega |Du|$

2.2 Le modèle TV : existence et unicité d'une solution BV

Notre objectif dans ce paragraphe est de traiter le problème de minimisation de la fonctionelle

$$\overline{E}(u) = \min_u \frac{1}{2} \int_\Omega |u_0 - Ru|^2 dx + \lambda \int_\Omega |Du| dx \tag{2.3}$$

quand l'opérateur R est connu.

2.2.1 Existence et unicité de la solution

Théorème 2.2 *On suppose que* $u_0 \in L^2(\Omega)$ *et que* $R : L^2(\Omega) \to L^2(\Omega)$, *est un opérateur linéaire continu et que* $R.1 \neq 0$. *Alors le problème de minimisation*

$$\inf_{u \in BV(\Omega)} \overline{E}(u) \tag{2.4}$$

admet une et une seule solution $u \in BV(\Omega)$

Démonstration On sépare la preuve en deux étapes

Etape 1 :*Existence :*

Soit $(u_n)_n$ une suite minimisante de (2.3), c'est-à-dire une suite de points qui vérifie

$$\text{pour tout } n \geq 0, \lim_{n \to +\infty} \overline{E}(u_n) = \inf_{u \in BV(\Omega)} \overline{E}(u) = M. \tag{2.5}$$

On a

$$\int_\Omega |u_0 - Ru_n|^2 dx \leq M \tag{2.6}$$

et

$$\int_\Omega |Du_n| \leq M.$$

Posons $w_n = \frac{1}{|\Omega|} \int_\Omega u_n dx$ et $v_n = u_n - w_n$, on remarque que $\int_\Omega v_n dx = 0$ et $Dv_n = Du_n$ et d'après l'inégalité de Poincaré on a :

$$\|v_n\|_2 \leq K \int_\Omega |Dv_n| dx = K \int_\Omega |Du_n| dx \leq K.M = M_1, \tag{2.7}$$

d'où

$$\|v_n\|_2 \leq M_1. \tag{2.8}$$

La suite (v_n) est donc bornée dans $L^2(\Omega)$. De plus, on a

$$\begin{aligned}|w_n|\|R1\|_2 = \|Rw_n\|_2 &= \|R(u_n - v_n)\|_2 \\ &\leq \|Ru_n - u_0\|_2 + \|Rv_n - u_0\|_2 \\ &\leq M + \|R\|\|v_n\|_2 + \|u_0\|_2 \\ &\leq M + \|R\|M_1 + \|u_0\|_2.\end{aligned}$$

Puisque $R1 \neq 0$, on en déduit que (w_n) est bornée dans $\mathbb{R}$. Donc la suite (u_n) est bornée dans $L^2(\Omega)$, et en particulier dans $L^1(\Omega)$ (d'après l'inégalité de Cauchy-Schwarz).

Par conséquent, il existe une sous-suite notée de la même façon (u_n) et une fonction $u \in BV(\Omega)$ telles que

$$u_n \overset{BV-w\star}{\rightharpoonup} u.$$

Par compacité de l'injection $BV(\Omega) \hookrightarrow L^1(\Omega)$ on peut de plus déduire que (quitte à reconsidérer une sous-suite)

$$u_n \to u \ , \ \text{ dans } \ L^1(\Omega).$$

On a alors

$$\int_\Omega |Du| \leq \liminf_{n\to+\infty} \int_\Omega |u_n|,$$

et

$$\overline{E}(u) \leq \liminf_{n\to+\infty} \overline{E}(u_n) = \inf_{u_n \in BV(\Omega)} \overline{E}(u_n) \leq \overline{E}(u)$$

Ce qui est équivalent à dire que u est un minimum de $\overline{E}(u)$. Le problème (2.4) admet donc une solution.

Etape 2 :*unicité*

Soient u et v deux minimums de $\overline{E}(u)$. On obtient facilement d'après la stricte convexité de $t \mapsto |t|$ que $Du = Dv$ ce qui implique que $u = v + c$. D'autre part, la fonction $w \mapsto \int_\Omega |w - u_0|^2 dx$ est strictement convexe, ce qui implique que $Ru = Rv$, donc $Rc = 0$. Puisque $R1 \neq 0$, on a $c = 0$.

□

2.2.2 Extension du modèle

Le modèle proposé précédemment correspond au cas d'une régularisation TV. Dans le chapitre (1), on a vu qu'une régularisation plus générale consiste à minimiser une fonctionnelle de la forme

$$E_\phi(u) = \frac{1}{2}\int_\Omega |u_0 - Ru|^2 dx + \lambda \int_\Omega \phi(|\nabla u|)dx \tag{2.9}$$

où ϕ est une fonction soumise à certaines contraintes assurant le maintien des contours et le lissage des zones homogènes en intensité. Notons que la fonctionnelle considérée dans le paragraphe précédent par l'équation (2.3) ne correspond pas exactement à la fonctionnelle (2.9) quand $\phi(t) = t$. En effet, le terme $\int_\Omega |Du|$ remplace le terme $\int_\Omega |\nabla u|$ non bien défini sur BV.
Comme nous l'avons souligné auparavant, l'espace naturel pour la minimisation de la fonctionnelle $E_\phi(u)$ est l'espace W défini en début de chapitre et qui est non réflexif. Comme dans le cas d'une régularisation de type TV, on peut reformuler ce problème dans le cas de l'espace $BV(\Omega)$. Le premier pas de cette reformulation consiste à étendre E_ϕ à l'espace BV de la façon suivante

$$\tilde{E}_\phi(u) = \begin{cases} E_\phi(u) \text{ si } u \in W, \\ +\infty \text{ si } u \in BV(\Omega)\backslash W. \end{cases}$$

Cette fonctionnelle n'est malheureusement pas semi-continue inférieurement pour la topologie $BV - w\star$. C'est pour cela qu'elle est souvent remplacée par son enveloppe semi-continue inférieure pour la topologie $BV - w\star$ (c'est-à-dire la plus grande fonctionnelle inférieure ou égale à $E_\phi(u)$ - étendue par $+\infty$ à tout $BV(\Omega)$- et $BV - w\star$ semi-continue inférieurement). On note $\overline{E}_\phi$ cette extension. Son expression dans le cas général nécessite l'introduction de quelques notions qui s'éloignent du sujet de ce mémoire (comme par exemple l'image par fonctions convexes d'une mesure de Radon). On renvoie au livre [5] pour cette expression dont nous n'avons pas besoin explicitement pour le déroulement du reste de ce mémoire.

On peut dès lors généraliser le résultat d'existence et d'unicité au cas où ϕ vérifie les hypothèses suivantes, très proches des conditions déduites dans le chapitre de modélisation (chap. 1) :

Condition 1 : ϕ est strictement convexe, croissante sur $\mathbb{R}^+$ avec $\phi(0) = 0$.

Condition 2 : $\lim_{s \to +\infty} \phi(s) = +\infty$

Condition 3 : Il existe deux constantes $c > 0$ et $b \geq 0$, telles que :

$$cs - b \leq \phi(s) \leq cs + b\,; \forall s \geq 0$$

Théorème 2.3 *Sous les hypothèses du théorème 2.2 et les conditions 1, 2 et 3 sur la fonction ϕ, la fonctionelle $\overline{E}_\phi$ admet un unique minimum dans $BV(\Omega)$.*

2.3 Méthode de l'approximation semi-quadratique

L'idée de la méthode d'approximation dite semi-quadratique est d'associer au problème de minimisation de $\overline{E}_\phi$, posé dans $BV(\Omega)$, une série de problèmes dans $W^{1,2}(\Omega)$ dont les solutions approchent le minimum de $\overline{E}_\phi$.

Plus précisément, pour tout $\epsilon > 0$, on considère le problème de minimisation

$$\min_{v \in W^{1,2}(\Omega)} E_\epsilon(v) = \frac{1}{2}\int_\Omega |u_0 - Rv|^2 dx + \lambda \int_\Omega \phi_\epsilon(|\nabla v|)dx, \tag{2.10}$$

où la fonction ϕ_ϵ est une approximation de ϕ donnée par

$$\phi_\epsilon(t) = \begin{cases} \dfrac{\phi'(\epsilon)}{2\epsilon}t^2 + \phi(\epsilon) - \dfrac{\epsilon\phi'(\epsilon)}{2} & \text{si } 0 \leq t \leq \epsilon, \\ \phi(t) & \text{si } \epsilon \leq t \leq \dfrac{1}{\epsilon}, \\ \dfrac{\epsilon\phi'(1/\epsilon)}{2}t^2 + \phi(1/\epsilon) - \dfrac{\phi'(1/\epsilon)}{2\epsilon} & \text{si } t \geq \frac{1}{\epsilon}, \end{cases} \tag{2.11}$$

On a le résultat suivant (voir [5])

Théorème 2.4 *Le problème de minimisation (2.10) admet une et une seule solution* v_ϵ*. De plus,* $v_\epsilon \in W^{1,2}(\Omega)$ *converge fortement dans* $L^1(\Omega)$ *vers l'élément* u *minimisant* $\overline{E}_\phi$ *dans* $BV(\Omega)$*. De plus,* $E_\epsilon(v_\epsilon)$ *converge vers* $\overline{E}_\phi(u)$*.*

L'équation d'Euler Lagrange associée au problème (2.10) est une EDP non linéaire difficile à résoudre. Une façon de reformuler ce problème afin de simplifier sa résolution a été proposée par Geman et Reynolds [26] est développée par la suite. Elle repose sur une technique de dualité. La proposition suivante est à la base de cette méthode :

Proposition 2.1 *Soit* $\phi : [0, +\infty[\to [0, +\infty[$*, telle que* $\phi(\sqrt{s})$ *soit strictement concave sur* $[0, +\infty[$*. Soient* L *et* M *définis comme suit :*

$$L = \lim_{s \to +\infty} \frac{\phi^{'}(s)}{2s} \text{ et } M = \lim_{s \to 0^+} \frac{\phi^{'}(s)}{2s}.$$

Alors :

1. *Il existe une fonction* $\psi :]L, M] \to [\alpha, \beta]$ *strictement convexe et décroissante telle que :*

$$\phi(u) = \inf_{L \leq b \leq M} (bu^2 + \psi(b))$$

   avec

$$\alpha = \lim_{u \to +\infty} \left(\phi(u) - u^2 \frac{\phi^{'}(u)}{2u} \right) \text{ et } \beta = \lim_{u \to 0^+} \phi(u).$$

2. *Pour tout* $u \geq 0$ *fixé, la valeur* b_u *pour la quelle l'inf est atteint :*

$$\inf_{L \leq b \leq M} (bu^2 + \psi(b)) = (b_u u^2 + \psi(b_u))$$

est unique et donnée par :

$$b = \frac{\phi'(u)}{2u}.$$

La démonstration de cette proposition est reléguée à la fin de ce chapitre.

L'avantage de cette proposition est qu'elle permet d'utiliser la fonction ψ pour reformuler le problème.

Nous avons introduit ci-dessous un tableau récapitulatif des fonctions ψ associées à des choix particulier de ϕ.

Nom	$\phi(u)$	$\psi(b)$	$\psi'(b)$
ϕ_{PM} Perona, Malik	$1 - \exp(-u^2)$	$b \log b - b + 1$	$\log b$
ϕ_{GM} German, McClure	$u^2/1 + u^2$	$(\sqrt{b} - 1)^2$	$1 - \frac{1}{\sqrt{b}}$
ϕ_{HL} Hebert, Leahy	$\log(1 + u^2)$	$b - \log b - 1$	$1 - \frac{1}{b}$
ϕ_{HS} Hyper Surfaces	$2\sqrt{1 + u^2} - 2$	$b + \frac{1}{4b} - \frac{5}{4}$	$1 - \frac{1}{4b^2}$

TABLE 2.1 – Fonctions de régularisation ϕ et fonctions ψ associées.

D'après Geman et Reynolds, on est ramené à calculer le minimum

$$\min_{u, L_\epsilon b \leq M_\epsilon} J_\epsilon(u, b) \tag{2.12}$$

où L_ϵ et M_ϵ sont associés à la fonction ϕ_ϵ comme dans la proposition (2.1) et

$$J_\epsilon(u, b) = \frac{1}{2} \int_\Omega |u_0 - Ru|^2 dx + \lambda \left(\int_\Omega b|\nabla u|^2 + \int_\Omega \psi_\epsilon(b) \right). \tag{2.13}$$

Cette fonctionelle est convexe en u et en b mais pas par rapport au couple (u, b). voir la proposition (2.1).

Ce résultat est à la base de l'agorithme de minimisation semi-quadratique qui consiste à minimiser J_ϵ de façon alternée par rapport à u et à b respectivement. Cet algorithme est exposé en détail au paragraphe 3.4 du chapitre suivant.

Preuve de la porposition 2.1

Notons $\theta(u) = \phi(\sqrt{u})$. D'après les conditions du théorème, $\theta(u)$ est strictement concave. Cette propriété a deux conséquences :

- d'une part, $\theta^{'}(u) = \dfrac{\phi^{'}(\sqrt{u})}{2\sqrt{u}}$ est strictement décroissante sur $[0, +\infty[$.
  Posons :
  $$L = \lim_{x \to +\infty} \frac{\phi^{'}(x)}{2x} \text{ et } M = \lim_{x \to 0^+} \frac{\phi^{'}(x)}{2x}.$$
  Si ces limites existent, alors $\theta^{'}$ est bijective et admet un inverse $(\theta^{'})^{-1} :]L, M] \to [0, +\infty[$.
- D'autre part, $\forall w, v : w \neq v,\ \theta(w) < \theta(v) + (w - v)\theta^{'}(v)$, ce qui revient à dire que θ est l'inf d'une famille de fonctions paramétrées par v :
  $$\theta(w) = \inf_v \left\{ \theta^{'}(v)w + \theta(v) - v\theta^{'}(v) \right\} \tag{2.14}$$

1. Posons pour $v \in [0, +\infty[$:
   $$\theta^{'}(v) = b \tag{2.15}$$
   ou de façon équivalente :
   $$v = (\theta^{'})^{-1}(b)$$
   Dans ce cas, (2.14) devient :
   $$\theta(w) = \inf_b \left\{ bw + \theta((\theta^{'})^{-1}(b)) - b(\theta^{'})^{-1}(b) \right\}$$
   En posant :
   $$\psi(b) = \theta((\theta^{'})^{-1}(b)) - b(\theta^{'})^{-1}(b)$$
   l'expression de $\theta(u)$ se met sous la forme :
   $$\theta(w) = \inf_b \{bw + \psi(b)\} \tag{2.16}$$

• La fonction ψ est à valeur sur l'intervalle définie par :

$$\psi(L) = \alpha = \lim_{v \to +\infty} \Big(\theta(v) - v\theta^{'}(v)\Big) \text{ et } \psi(M) = \beta = \lim_{v \to 0^+} \Big(\theta(v) - v\theta^{'}(v)\Big)$$

soit, en posant $v = u^2$:

$$\alpha = \lim_{u \to +\infty} \left(\phi(u) - u^2 \frac{\phi^{'}(u)}{2u}\right) \text{ et } \beta = \lim_{u \to 0^+} \left(\phi(u) - u^2 \frac{\phi^{'}(u)}{2u}\right)$$

soit encore

$$\alpha = \lim_{u \to +\infty} \left(\phi(u) - u^2 \frac{\phi^{'}(u)}{2u}\right) \text{ et } \beta = \lim_{u \to 0^+} \phi(u)$$

• Un calcul élémentaire montre que $\psi^{'}(b) = -(\theta^{'})^{-1}(b)$:

$$\begin{aligned}
\psi^{'}(b) &= [(\theta^{'})^{-1}(b)]^{'} \theta^{'}((\theta^{'})^{-1}(b)) - (\theta^{'})^{-1}(b) - b[(\theta^{'})^{-1}(b)]^{'} \\
&= [(\theta^{'})^{-1}(b)]^{'} b - (\theta^{'})^{-1}(b) - b[(\theta^{'})^{-1}(b)]^{'} \\
&= -(\theta^{'})^{-1}(b).
\end{aligned}$$

Comme $(\theta^{'})^{-1}$ est à valeur sur $]0, +\infty[$, ceci montre que ψ est strictement décroissante sur $]L, M]$.

• D'autre part, il est facile de montrer que $\psi^{''}(b) = \dfrac{-1}{\theta''(v)}$. En effet, on a par définition :

$$\begin{aligned}
\psi^{''}(b) &= \lim_{c \to 0} \frac{\psi^{'}(b+c) - \psi^{'}(b)}{c} \\
&= \lim_{c \to 0} \frac{\psi^{'}(b+c) - \psi^{'}(b)}{b+c-b} \\
&= \lim_{c \to 0} \frac{-(\theta^{'})^{-1}(b+c) + (\theta^{'})^{-1}(b)}{\theta'[(\theta')^{-1}(b+c)] - \theta'[(\theta')^{-1}(b)]}.
\end{aligned}$$

En posant $e = (\theta^{'})^{-1}(b+c)$ et en utilisant (2.15), il vient donc :

$$\begin{aligned}
\psi^{''}(b) &= \lim_{e \to v} -\frac{v - e}{\theta'[v] - \theta'[e]} \\
&= -\frac{1}{\theta''(v)}.
\end{aligned}$$

Or, dans les hypothèses du théorème, $\theta^{''}(v)$ est strictement négative. La fonction ψ est donc strictement convexe.

• Enfin, en posant $w = u^2$, l'équation (2.16) devient :

$$\theta(u^2) = \phi(u) = \inf_b \{bu^2 + \psi(b)\}.$$

2. D'après (1), il est évident qu'à w fixé, l'inf est atteint pour $v = w$, c'est-à-dire pour $b = \theta^{'}(w)$. Avec $w = v^2$, nous avons donc bien $b_u = \dfrac{\phi^{'}(u)}{2u}$. On prolongera cette expression par continuité en 0.

$\square$

Chapitre 3

Le cas d'une image discrète : de la théorie à l'implémentation

Dans ce chapitre on se place dans le cas d'une image discrète. Nous allons reprendre les modèles exposés dans le chapitre 1 et étudiés dans le cadre continue en chapitre 2. Le but ici est non seulement de les étudier mathématiquement dans le cas discret, mais de les mettre en oeuvre numériquement en les implémentant dans des codes écrits en langage C ou avec le logiciel matlab. Ce chapitre constitue le travail central de ce mémoire.

Nous allons essentiellement exposer deux algorithmes :

– L'algorithme de projection de Chambolle qu'on utilisera essentiellement dans le cas de débruitage ($R = I$).
– L'agorithme de minimisation semi-quadratique, basée sur les résultats du paragraphe (2.3), du chapitre (2).

3.1 Le problème discret

Dans le domaine de l'imagerie numérique qui nous intéresse ici, les appareils de photographie par exemple échantillonne le monde réel sur une grille par l'intermédiaire de capteurs. L'image obtenue est alors un ensemble de cellules de cette grille, appelées *pixels* dans le plan (2D) et *voxels* dans l'espace (3D), chaque cellule contenant une partie de l'information de l'image globale. C'est sur la base de cette représentation discrète que nous sommes amenés à travailler pour extraire les constituants de l'image. Les images discrètes ne sont qu'un modèle pratique pour présenter des "vues" de l'espace continu dans la mémoire d'un calculateur.

On va se placer dans le cas d'image 2D, comportant N pixels dans la direction horizontale et M pixels dans la direction verticale.

Définition 3.1 *Une image numérique en niveau de gris est une matrice* $(u_{i,j})_{1\leq i\leq N, 1\leq j\leq M}$ *où à chaque pixel* (i, j)*, intersection de la ligne* i *et la colonne* j*, on fait correspondre un niveau de gris* $u_{i,j} \in \{0, ..., 255\}$.

En pratique, on modélise une image discrète par un vecteur $u \in X = \mathbb{R}^{N\times M}$ (N et M sont respectivement les nombres de pixels dans les deux directions horizontale et verticale). Les contraintes $0 \leq u_{i,j} \leq 255$ seront souvent ignorées mathématiquement comme dans la plupart des articles dans la littérature. Néanmoins, ces contraintes seront prises en compte lors de l'implémentation.

On note X l'espace euclidien $\mathbb{R}^{N\times N}$ et $Y = X \times X$. On munit X du produit scalaire usuel :

$$\langle u, v\rangle_X = \sum_{1\leq i,j\leq N} u_{i,j} v_{i,j} \tag{3.1}$$

et de la norme associée :$\|.\|$.

Nous introduisons une version discrète de l'opérateur gradient. Si $u \in X$, le gradient ∇u est un vecteur de Y donné par :

$$(\nabla u)_{i,j} = \left((\nabla u)^1_{i,j}, (\nabla u)^2_{i,j}\right) \tag{3.2}$$

avec comme choix de discrétisation ici

$$(\nabla u)^1_{i,j} = \begin{cases} u_{i+1,j} - u_{i,j} & \text{si } i < N \\ 0 & \text{si } i = N. \end{cases}$$

$$(\nabla u)^2_{i,j} = \begin{cases} u_{i,j+1} - u_{i,j} & \text{si } j < N; \\ 0 & \text{si } j = N. \end{cases}$$

La variation totale discrète est alors donnée par

$$J(u) = \sum_{1\leq i,j\leq N} |(\nabla u)_{i,j}|. \tag{3.3}$$

où

$$|(\nabla u)_{i,j}| = \sqrt{|(\nabla u)^1_{i,j}|^2 + |(\nabla u)^2_{i,j}|^2}$$

On introduit également une version discrète de l'opérateur de divergence. On le définit par analogie avec le cadre continu en posant

$$\mathrm{div} = -\nabla^*$$

où ∇^* est l'opérateur adjoint de ∇, c'est-à-dire celui vérifiant

$$\forall p \in Y, \forall u \in X : \langle -\mathrm{div} p, u\rangle_X = \langle p, \nabla u\rangle_Y = \langle p^1, \nabla^1 u\rangle_X + \langle p^2, \nabla^2 u\rangle_X$$

On peut alors vérifier que :

$$(\mathrm{div} p)_{i,j} = (\mathrm{div} p)^1_{i,j} + (\mathrm{div} p)^2_{i,j},$$

où

$$(\mathrm{div} p)^1_{i,j} = \begin{cases} p^1_{i,j} - p^1_{i-1,j} & \text{si } 1 < i < N \\ p^1_{i,j} & \text{si } i = 1 \\ -p^1_{i-1,j} & \text{si } i = N. \end{cases}$$

$$(\mathrm{div} p)^2_{i,j} = \begin{cases} p^2_{i,j} - p^2_{i,j-1} & \text{si } 1 < i < N \\ p^2_{i,j} & \text{si } j = 1 \\ -p^2_{i,j-1} & \text{si } j = N. \end{cases}$$

La version discrète du problème de minimisation (2.1) quand $\phi(t) = t$ s'écrit alors

$$\min_{u \in X} J(u) = \frac{1}{2}\|Ru - u_0\|_2^2 + \lambda J_{TV}(u) \tag{3.4}$$

où

$$J_{TV}(u) = \sum_{i=1}^{N}\sum_{j=1}^{M} |(\nabla u)_{i,j}|.$$

Ici R est considérée comme un endomorphisme de X. On conviendra dans toute la suite que le signe $\int_\Omega$ est à comprendre au sens discret, c'est-à-dire

$$\int_\Omega f = \sum_{o=1}^{N}\sum_{j=1}^{M} f_{i,j}.$$

(Une telle notation évitera de mettre des doubles sommes partout).

3.2 Condition d'optimalité

La fonction J donnée par (3.4) est convexe. Néanmoins, comme nous allons le voir, elle n'est pas différentiable en certains u (ceux pour lesquelles il existe au moins un pixel (i, j) tel que $(\nabla u)_{i,j} = 0$).

Afin d'écrire la condition d'optimalité d'ordre 1 associée à cette fonctionelle, nous allons introduire la notion de sous-différentiel.

Définition 3.2 *(Sous-differentiel) Soit $f : X \to \mathbb{R}$ une fonction convexe. Le sous-differentiel de f en $x \in X$ est l'ensemble $\partial f(x)$ des éléments $w \in X$ tels que*

$$\forall y \in X : \; f(y) \geq f(x) + \langle w, y - x \rangle. \tag{3.5}$$

Les éléments de $\partial f(x)$ sont appelés les sous-gradients de f en x.

Notons que le sous-différentiel en un point n'est pas un ensemble vide. Il est réduit au singleton $\{\nabla f(x)\}$ quand f est différentiable. On renvoie à [19] par exemple pour ses propriétés.
Le lemme suivant est évident

Lemme 3.2.1 *Un élément $u \in X$ réalise le minimum de J si et seulement si*

$$0 \in \partial J(u). \tag{3.6}$$

Afin d'écrire explicitement la condition d'optimalité associée au problème (3.4), on a besoin de calculer la sous-différentiel de J en tout point. La proposition suivante est démontrée dans [**?**]

Proposition 3.1 $0 \in \partial J(u)$ *si et seulement si il existe $p \in X^2$ tel que*

$$K^\star Ru - R^\star u_0 + \lambda \mathrm{div} p = 0, \tag{3.7}$$

et p vérifiant pour tous i, j
- $p_{ij} = -\dfrac{(\nabla u)_{i,j}}{|(\nabla u)_{i,j}|}$ *si* $(\nabla u)_{i,j} \neq 0$,
- $|p_{ij}| \leq 1$ *si* $(\nabla u)_{i,j} = 0$.

Notons que l'équation (3.7) est l'équivalent discret de l'équation d'Euler-Lagrange associée au problème continue.

3.3 Algorithme de Chambolle dans le cas du denosing

Dans le cas du problème de denoising ($R = I$) il existe une caractérisation de la solution due à Chambolle [14]. Afin de donner cette caractérisation, considérons l'ensemble convexe suivant

$$G = \{\text{div}g \mid g \in X,\ |g_{ij}| \leq 1,\ \forall\, i, j\}$$

Le résultat suivant donne la caractérisation attendu de la solution.

Proposition 3.2 *On suppose que $R = I$. La solution de (3.4) est donnée par*

$$u = u_0 - P_{\lambda G}(u_0) \tag{3.8}$$

où $P_{\lambda G}$ est la projection orthogonale sur le convexe λG.

Preuve : Soit

$$B_1 = \{p \in X^2 \mid |p_{i,j}| \leq 1\}.$$

On a quand $R = I$

$$J(u) = \frac{1}{2}\int_\Omega (u - u_0)^2 + \lambda \int_\Omega |\nabla u|$$

J est continue et strictement convexe. De plus on a

$$\lim_{\|u\| \mapsto +\infty} J(u) = +\infty.$$

Donc J admet un unique minimum sur X réalisé en un point qu'on notera $u^\star$.

La condition d'optimalité de la proposition 3.1 donne

$$u^\star = u_0 - \lambda \text{div} p^\star \text{ avec } |p^\star| \leq 1 \text{ et } p^\star.\nabla u^\star = |\nabla u^\star|.$$

On a pour tout $p \in G$,

$$\begin{aligned} J(u) &\geq \frac{1}{2}\int_\Omega (u-u_0)^2 - \lambda \int_\Omega p.\nabla u \\ &\geq \frac{1}{2}\int_\Omega (u-u_0)^2 + \lambda \int_\Omega u\text{div}p \\ &\geq \frac{1}{2}\int_\Omega (u-u_0+\lambda\text{div}p)^2 - \frac{\lambda^2}{2}\int_\Omega (\text{div}p)^2 + \lambda \int_\Omega \text{div}pu_0 \\ &= \frac{1}{2}\int_\Omega (u-u_0+\lambda\text{div}p)^2 - \frac{1}{2}\int_\Omega (u_0-\lambda\text{div}p)^2 + \frac{1}{2}\int_\Omega |u_0|^2 \\ &\geq -\frac{1}{2}\int_\Omega (u_0-\lambda\text{div}p)^2 + \frac{1}{2}\int_\Omega |u_0|^2. \end{aligned}$$

En passant au supremum sur p à droite on en déduit que

$$J(u) \geq -\frac{1}{2}\int_\Omega (u_0 - P_{\lambda G}u_0)^2 + \frac{1}{2}\int_\Omega |u_0|^2 dx.$$

où $P_{\lambda C}u_0$ est la projection de u_0 sur λG. Par ailleurs, on a

$$\begin{aligned} J(u^\star) &= \frac{1}{2}\int_\Omega (u^\star-u_0)^2 - \lambda \int_\Omega p^\star.\nabla u^\star \\ &= -\frac{1}{2}\int_\Omega (u_0-\lambda\text{div}p^\star)^2 + \frac{1}{2}\int_\Omega |u_0|^2. \\ &\leq -\frac{1}{2}\int_\Omega (u_0-P_{\lambda G}u_0)^2 + \frac{1}{2}\int_\Omega |u_0|^2, \end{aligned}$$

avec égalité si et seulement si $\lambda\text{div}p^\star = P_{\lambda G}u_0$. Il en résulte des deux inégalités que

$$J(u^\star) = -\frac{1}{2}\int_\Omega (u_0 - P_{\lambda G}u_0)^2 + \frac{1}{2}\int_\Omega |u_0|^2,$$

et que $\lambda\text{div}p^\star = P_{\lambda G}u_0$. D'où le résultat.

□

Le résultat 3.2 permet de réduire le calcul de la solution u au calcul de la projection de u_0 sur le convexe λG. Chambolle a proposé le schéma de point fixe suivant

– $p^{(0)} = 0$.

– pour $n \geq 0$,

$$p_{i,j}^{n+1} = \frac{p_{i,j}^n + \tau(\nabla(div(p^n) - u_0/\lambda))_{i,j}}{1 + \tau|(\nabla(div(p^n) - u_0/\lambda))_{i,j}|}$$

Ici $\tau > 0$ désigne un paramètre réel fixé.

Le théorème suivant donne une condition suffisante pour que l'algorithme converge :

Théorème 3.1 *Supposons que le paramètre τ vérifie $0 < \tau \leq 1/8$. Alors, $\lambda div(p^n)$ converge vers $P_{\lambda G}(u_0)$ quand $n \to +\infty$*

La solution du problème (3.4) est donnée par :

$$u = u_0 - \lambda \mathrm{div} p^\infty, \tag{3.9}$$

où

$$p^\infty = \lim_{n \to +\infty} p^n$$

Nous avons implémenté cet algorithme dans un code en langage C dont le listing est joint en annexe à ce rapport.

Cet algorithme a été étendu au cas $R \neq I$ (voir [35]). Néanmoins, dans ce cas, l'algorithme obtenu est lent en terme de temps de calcul notamment quand R^*R possède des valeurs propres petites. Nous avons préférée utiliser la méthode de minimisation semi-quadratique, exposée dans le paragraphe suivant.

3.4 Méthode de minimisation semi-quadratique

Dans le chapitre (2), paragraphe (2.3), on avait montré que la solution u du problème (2.10) peut être approchée par u_ϵ où le couple (u_ϵ, b_ϵ) réalise le minimum de

$$J_\epsilon(u,b) = \int_\Omega |u_0 - Ru|^2 dx + \lambda \left(\int_\Omega b|\nabla u|^2 + \int_\Omega \psi_\epsilon(b) \right) \tag{3.10}$$

Le schéma suivant appelé ARTUR est utilisé pour résoudre ce problème en pratique

- (u^0, b^0) donnée
- pour tout $n \geq 0$,
  - $u_\epsilon^{n+1} = \min J_\epsilon(u, b^n)$. Quand ce problème est convexe, il est équivalent au problème suivant :

$$\begin{cases} R^*Ru - \lambda \mathrm{div}(b\nabla u) = R^*u_0 & \text{dans } \Omega, \\ b^n \dfrac{\partial u}{\partial N} = 0 & \text{sur } \partial\Omega. \end{cases}$$

  - $b_\epsilon^{n+1} = \min J_\epsilon(u_\epsilon^{n+1}, b)$. Le minimum est atteint pour

$$b^{n+1} = \frac{\varphi'(|\nabla u_\epsilon^{n+1}|)}{2|\nabla u_\epsilon^{n+1}|}.$$

La limite $(u_\epsilon^\infty, b^\infty)$ est la solution.

Remarque 3.4.1 *On peut voir la suite $b^n(x)$ comme un indicateur de contours. Si φ satisfait les hypothèses de préservation de contours*

- *si $b^n(x) = 0$, alors x appartient à un contour.*
- *si $b^n(x) = 1$, alors x appartient à une zone homogène.*

Nous allons détailler maintenant la résolution du premier problème en u^n, en résolvant l'équation

$$R^*Ru - \lambda \mathrm{div}(b\nabla u) = R^*u_0 \tag{3.11}$$

On a la discrétisation suivante issue de la formulation variationnelle pour cette dernière équation : $\forall \varphi \in X$

$$\int_\Omega Ru.R\varphi + \lambda \int_\Omega b\nabla u.\nabla \varphi = \int_\Omega u_0.R\varphi$$

Soit la base $(\varphi^{(k,m)})_{1\leq k\leq N,\, 1\leq m\leq N}$ de X définie comme suit

$$\varphi^{(k,m)}_{(i,j)} = \begin{cases} 1 & \text{si } (k,m) = (i,j) \\ 0 & \text{sinon .} \end{cases}$$

En utilisant les éléments de cette base comme fonctions tests dans la formulation précédente, on obtient : pour tous $1 \leq k \leq N$ et $1 \leq m \leq M$,

$$\int_\Omega Ru.R\varphi^{(k,m)} + \lambda \int_\Omega b\nabla u.\nabla \varphi^{(k,m)} = \int_\Omega u_0.R\varphi^{(k,m)}$$

On a $u = \Sigma_{i,j=1}^{N,M} u_{i,j}\varphi^{(i,j)}$ et $b\nabla u = \Sigma_{i,j=1}^{N,M} (\nabla u)_{i,j} b_{i,j}\varphi^{(i,j)} = \Sigma_{i,j=1}^{N,M} u_{i,j} b_{i,j} \nabla\varphi^{(i,j)}$.

En les injectant dans l'équation précédente on obtient : pour tous $1 \leq k \leq N$ et $1 \leq m \leq M$,

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\int_\Omega R\varphi^{(i,j)}.R\varphi^{(k,m)}\right) u_{i,j} + \lambda \sum_{i=1}^{N}\sum_{j=1}^{M}\left(b_{i,j}\int_\Omega \varphi^{(i,j)}\nabla\varphi^{(k,m)}\right)(\nabla u_{i,j})_{i,j}$$
$$= \textstyle\sum_{i=1}^{N}\sum_{j=1}^{M}\int_\Omega \varphi^{(i,j)} R\varphi^{(k,m)}(u_0)_{i,j}$$

ou encore,

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\int_\Omega R\varphi^{(i,j)}.R\varphi^{(k,m)}\right) u_{i,j} + \lambda \sum_{i=1}^{N}\sum_{j=1}^{M}\left(b_{i,j}\int_\Omega \nabla\varphi^{(i,j)}\nabla\varphi^{(k,m)}\right) u_{i,j}$$
$$= \textstyle\sum_{i=1}^{N}\sum_{j=1}^{M}\int_\Omega \varphi^{(i,j)} R\varphi^{(k,m)}(u_0)_{i,j}$$

Finalement on obtient,

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\int_\Omega R\varphi^{(i,j)}.R\varphi^{(k,m)} + \lambda b_{i,j}\int_\Omega \nabla\varphi^{(i,j)}\nabla\varphi^{(k,m)}\right) u_{i,j}$$
$$= \textstyle\sum_{i=1}^{N}\sum_{j=1}^{M}\int_\Omega \varphi^{(i,j)} R\varphi^{(k,m)}(u_0)_{i,j}$$

En utilisant les formules

$$\partial_x \varphi_{(i,j)}^{(k,m)} = \begin{cases} -1 & \text{si } (k,m) = (i,j) \\ 1 & \text{si } (k,m) = (i-1,j) \\ 0 & \text{sinon ,} \end{cases}$$

$$\partial_y \varphi_{(i,j)}^{(k,m)} = \begin{cases} -1 & \text{si } (k,m) = (i,j) \\ 1 & \text{si } (k,m) = (i,j-1) \\ 0 & \text{sinon .} \end{cases}$$

on montre que l'équation (3.11) se ramène finalement à un système linéaire

$$AU = B,$$

avec A une matrice symétrique définie positive. Cette matrice est creuse. En pratique, nous résolvons ce système avec une méthode itérative de type Gradient Conjugué (voir annexe). Notons que l'un des avantages de cette méthode est la non nécessité de stoker en mémoire la matrice A ; il suffit de programmer une subroutine ayant à l'entrée le vecteur U renvoie à la sortie le vecteur AU.

On a aussi implémenté cette méthode dans un programme écrit en langage C (voir listings en Annnexe).

Nous montrons ici des résultats à partir de simulations que nous avons effectuées en utilisant une image de taille (118*121) type ".tif" afin de vérifier la validité de cet algorithme de semi-quadratique. Cette image test est construite par un dessin à la main avec le logiciel Paint. Voir la figure (3.1)

Figure 3.1 – Image originale de taille 118*121

Cette image a été dégradée par un noyau de disque de rayon égal à 3. Le résultat du floutage est illustré dans la figure (3.2)

La régularisation utilisée est la régularisation avec prise en compte de discontinuités avec la fonction $\phi(t) = 2\sqrt{1+t^2} - 2$. La convergence est obtenue pour cette image en moins de 10 itérations avec un ISNR égale à 7.70 Le résultat est illustré par la figure (3.3)

L'image des contours (la variable b) est aussi présentée dans la figure (3.4)

FIGURE 3.2 – Image floutée avce un noyau de rayon 3

FIGURE 3.3 – Résultat de la déconvolution par régularisation semi-quadratique

FIGURE 3.4 – Image des contours (la variable b)

En regardant l'évolution des itérations en fonction de l'erreur, on remarque la convergence très rapide de cet algorithme à partir de la quatrième itération. Voir figure (3.5)

Maintenant on va appliquer le même algorithme sur une image qui contient plus d'information (encore celle de Barbara) et voir le résultat qui est présenté dans la figure (3.6)

FIGURE 3.5 – graphe des itérations en fonction de l'erreur

FIGURE 3.6 – Contours (la variable b) pour l'image de Barbara

On constate qu'on peut extraire toutes les lignes de niveau que contient cette dernière. Ce qui prouve encore une fois l'éfficacité de cette méthode qui pourra avoir beaucoups d'applications dans de nombreux domaines.

À travers ce paragraphe, nous allons mener diverses comparaisons entre les convergences des algorithmes de Chambolle et celui de la semi-quadratique avec la fonction $\phi(t) = 2\sqrt{1+t^2} - 2$. Dans les deux cas, on suppose uniquement la présence d'un bruit gaussien d'écart type 20 et de moyenne 0.008 qui est appliqué sur l'image de façon additif. l'opérateur de dégradation R est égale à l'identité (pas de floutage).

L'image originale de test est toujour celle de Barbara (512x512) en niveau de gris. voir Figure (3.7)

FIGURE 3.7 – l'image de Barbara originale

Et, après avoir appliqué le bruit en question, on obtient la figure (3.8) :

FIGURE 3.8 – Barbara modifiée avec un bruit gaussien de $\sigma = 20$ et $\mu = 0.008$

Le résultat du test par l'algorithme de Chambolle est illustré par la figure (3.9)

FIGURE 3.9 – Image de Barbara restaurée avec algorithme de Chambolle

Pour l'algorithme de la semi-quadratique, l'image (3.10) représente le résultat du test

FIGURE 3.10 – Image de Barbara restaurée avec algorithme de la semi-quadratique

On remarque que l'image restaurée avec la régularisation semi-quadratique présente moins de bruit dans les zones homogènes. Quant aux résultats de l'algorithme de Chambolle on constate la disparition du bruit avec un lissage considérable de l'image (effet cartoon).

Nous pouvons aussi comparer leur vitesse de convergence, par rapport à leur nombre d'itérations, voir le graphe suivant

FIGURE 3.11 – Itérations en fonction de l'erreur pour l'algorithme de Chambolle

FIGURE 3.12 – Itérations en fonction de l'erreur pour la méthode semi-quadratique

On peut voir que les deux algorithmes convergent vers la solution originale. Ce pendant, l'interêt de l'algorithme de la semi-quadratique est sa rapidité de convergence ; il atteint son but à partir de l'itération 10, alors qu'il faut à l'algorithme de Chambolle plus de 20 itérations pour y parvenir. L'algorithme de Chambolle converge avec un ISNR positif égale à 5.22 pour un paramètre de lissage $\lambda = 20$. Pour l'algorithme de la semi-quadratique, il converge avec un ISNR égale à 40.76

D'une autre part, on relève que le choix du paramètre λ influe très sensiblement sur la qualité des résultats, c'est pourquoi il est nécessaire de les ajuster précisément. On remarque que ce paramètre varie d'une scène à l'autre et ce n'est que par essais et erreurs qu'on arrive à avoir de meilleurs résultats. Le graphe (3.13) représente la variation du paramètre λ en fonction de l'ISNR dans le cas de l'algorithme de Chambolle, et il renforce les remarques établies.

Figure 3.13 – Variation de Lambda en fonction de ISNR

On conclut finalement, que la méthode de la semi-quadratique réalise un bon compromis entre temps de calcule et qualité d'image reconstruite par rapport à celle de Chambolle qui lisse trop les contours. De plus la semi-quadratique est plus générale, au sens où elle peut être mise en œuvre pour une variété importante de fonctions de potentiels ϕ.

Annexe A

méthode du gradient conjugué en bref

Notre but ici est d'exposer une méthode pour la résolution des problèmes du type :

$$(\mathcal{P}),\ \min f(x), x \in \mathbb{R}^* \tag{A.1}$$

Quand la dimension n est supérieure à 1. On rappelle que dans ce cas la condition d'optimalité d'ordre 1 s'écrit

$$\nabla f(x^*) = 0. \tag{A.2}$$

Le principe général des méthodes numériques d'optimisation consiste à construire une suite $x^{(0)}, ..., x^{(k)}, ...$ destinée à converger vers la solution x^*, de préférence rapidement. le plus souvent, le déplacement de $x^{(k)}$ à $x^{(k+1)}$ dans cette suite se fait en deux étapes

• On choisit d'abord une direction de descente d_k (c'est-à-dire dans laquelle f va décroître),

• On choisit ensuite la taille du pas de ce déplacement, et donc $x^{(k+1)} =$

$x^{(k)} + \alpha_k d_k$, en "minimisant" f sur la demi droite $x^{(k)} + \alpha d_k, \alpha \geq 0$.
Cette dernière étape révèle toute l'importance des méthodes d'optimisation monodimensionnelles. Il est toutefois important de souligner que puisque le but est de trouver l'optimum de f, on peut se contenter d'une recherche (dite "économique") d'un pas convenable, assurant en général une décroissance de f, sans être forcément de taille optimale. En effet, la recherche d'un pas optimal α_k à chaque itération peut s'avérer très coûteuse car elle nécessite beaucoup d'évaluations de f et de ses dérivées.
Dans les méthodes du gradient, on part d'un point x^0 et on construit la suite $x^{(0)}, ..., x^{(k)}, ...$ de la façon décrite précédemment en choisissant une direction du déplacement opposée au gradient $d_k = -\nabla f(x^{(k)})$. Ce choix est bien évidemment motivé par le fait qu'un petit déplacement dans cette direction entraine à priori la décroissance de f puisque pour $t > 0$ suffisamment petit on a

$$f(x^{(k)} + td_k) - f(x^{(k)}) = td_k \nabla f(x^{(k)}) + o(t) = -t\|\nabla f(x^{(k)})\|^2 + o(t) < 0.$$

Supposons maintenant que la fonction f est quadratique de la forme

$$f(x) = \frac{1}{2} x^T A x - b^T x \tag{A.3}$$

L'idée principale de l'algorithme de gradient conjugué consiste à construire d'une part une famille de vecteurs $d_0, ..., d_{n-1}$, deux à deux conjuguées par rapport à la matrice A (i.e. $d_k^T A d_m = \delta_{k,m}$ pour tous k et m, et d'autre part de calculer les points $x^{(k)}$ comme solutions des problèmes d'optimisation

$$\min_{x \in W_k} f(x), \text{ où } W_k = x^{(0)} + vect\{d_0, ..., d_{n-1}\} \tag{A.4}$$

Ainsi, pour tout $k \geq 0$, $x^{(k)}$ s'écrit sous la forme

$$x^{(k)} = x^{(0)} + \Sigma_{i=1}^{k-1} \lambda_i d_i. \tag{A.5}$$

Le dernier élément x_n minimise f sur W_n qui est forcément $\mathbb{R}^n$ tout entier. Plus exactement, les couples $(x^{(k)}, d_k)$ sont calculés par récurrence de la manière suivante :

$d_0 = -\nabla f(x^{(0)}, x^{(k+1)} = x^{(k)} + \lambda_k d_k, d_{k+1} = -\nabla f(x^{k+1}) + \beta_k d_k, pour k \geq\ 0,$

où le coefficient λ_k réalise le minimum de $f(\lambda_k + d_k)$, c'est-à-dire

$$\lambda = -\frac{\nabla f(x^{(k)})^T . d_k}{d_k^T A d_k}, \tag{A.6}$$

tandis que le coefficient β_k est choisi pour assurer la conjugaison de d_{k+1} et d_k par rapport à A, i.e.

$$\beta_k = \frac{\nabla f(x^{(k+1)})^T A d_k}{d_k^T A d_k}. \tag{A.7}$$

On peut alors montrer que les directions $d_k, k = 0, 1, ...$ sont deux à deux conjuguées par rapport à A et que les gradients $\nabla f(x_k), k = 0, 1, ...$ sont deux à deux orthogonaux. On montre aussi que si l'optimum n'est pas atteint à la k-ème itération ($\nabla f(x^{(k)}) \neq 0$), alors

$$\lambda = \frac{\|\nabla f(x^{(k)})\|^2}{d_k^T A d_k}, \beta_k = \frac{\|\nabla f(x^{(k+1)})\|^2}{\|\nabla f(x^{(k)})\|^2}. \tag{A.8}$$

Annexe A

Listings des programmes

```
// *************** projection_chambolle *****************
// Last updated 25-06-2010

// Chambolle's original method (2004).
// Convergence can be proved for tau <= .125.
// optimal performance occurs for lambda between 15~20

// \div : divergence,  \g : gradient
//------------------------------------------------------------------------
// Input variables
//------------------------------------------------------------------------
// px,py:         Initial guess.
// u:             Noisy image
// lambda:        Constant fidelity parameter.
// nmax:          Maximum number of iterations
// tol:           Convergence tolerance (stop criterion)
```

```
//-------------------------------------------------------------------------
// Output variables
//-------------------------------------------------------------------------
// pr:           Primal variable, numerical solution - restored image
// itr:          number of iteration
// ISNR:          measure quality of restoration
//--------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int  ind_2d1d(int i,  int j, int nx, int ny);
void divergence(int nx, int ny, float px[], float py[], float div[] );
void gradientX(int nx, int ny, float u[], float gx[]);
void gradientY(int nx, int ny, float u[], float gy[]);
void projection_chambo(int nx, int ny, float lambda, float u[], float pr[],
                       float tau, float tol, int nmax);
float ISNR( int n, float u[], float v[], float w[]);
main (){
   int nx, ny, taille, nmax=110, k;
   float lambda = 20.  , tau=0.125, tol = 1e-4;
      float *pr, *u, *res;
        FILE *orig;
        orig = fopen("orig.txt","r");
        fscanf(orig,"%d\n",&nx);
        fscanf(orig,"%d\n",&ny);
         printf("%d, %d\n",nx ,ny);
```

```
          taille = nx*ny;
          u   = malloc(taille*sizeof(float));
          res = malloc(taille*sizeof(float));
          pr  = malloc(taille*sizeof(float));
     for (k = 0; k< nx*ny; k++)
         fscanf(orig,"%f\t", &u[k]);
          fclose(orig);
          orig = fopen("bruitee.txt","r");
       for (k = 0; k< nx*ny; k++)
         {fscanf(orig,"%f\t",&res[k]);}
          fclose(orig);
projection_chambo(nx, ny, lambda, res, pr, tau, tol, nmax);
printf("ISNR= %6.2e \n",ISNR(nx*ny, u, res, pr));
                  FILE *orag;
                  orag=fopen("orag.txt","w");
              for (k = 0; k< nx*ny; k++)
                     fprintf(orag,"%6.2f\t", pr[k]);
                  fclose(orag);
return (0);   }
 int ind_2d1d(int i,  int j, int nx, int ny)
           { int ind ;
             ind = j*nx + i;
             return (ind) ; }
           void ind_1d2d(int ind, int i, int j, int nx, int ny)
           { j = ind/nx;
             i = ind - j*nx; }
```

```
void divergence(int nx, int ny, float px[], float py[], float div[] )
            {
//           les tableaux px, py, div sont des tableaux monodimensionnels
//           de la taille de l'image, c'est ‡ dire de taille nx*ny
//           Les stockage est fait ligne par ligne
              int i, j, k;
              for(i=1; i<nx-1; i++)
               {  for(j=0; j<ny; j++)
                  { k = ind_2d1d(i, j, nx, ny);
                    div[k] = px[k]- px[k-1];}
               }
               i = 0;
for(j=0; j<ny; j++){ k = ind_2d1d(i, j, nx, ny);
                      div[k] = px[k];
                    }
               i = nx -1;
for(j=0; j<ny; j++){  k = ind_2d1d(i, j, nx, ny);
                       div[k] = -px[k-1];
                    }
               for(j=1; j<ny-1; j++)
                {   for(i=0; i<nx; i++)
                    {  k = ind_2d1d(i, j, nx, ny);
                       div[k] += py[k]- py[k-nx];
                    }
                }
               j = 0;
```

```
for(i=0; i<nx; i++){  k = ind_2d1d(i, j, nx, ny);
                      div[k] += py[k];
                   }
              j = ny-1;
for(i=0; i< nx; i++){ k = ind_2d1d(i, j, nx, ny);
                      div[k] -= py[k-nx];
                    }
            }
void  gradientX(int nx, int ny, float u[], float gx[])
             {
              int i, j, k;
    for(i=0; i<nx-1; i++)
        { for(j=0; j<ny; j++)
                 {  k = ind_2d1d(i, j, nx, ny);
                    gx[k] = u[k+1]- u[k];
                 }
        }
              i = nx-1;
    for(j=0; j<ny; j++) {  k = ind_2d1d(i, j, nx, ny);
                           gx[k] = 0;
                        }
             }
void gradientY(int nx, int ny, float u[], float gy[])
              {
               int i, j, k;
               for(j=0; j<ny-1; j++)
```

```
                  {  for(i=0; i<nx; i++)
                   {  k = ind_2d1d(i, j, nx, ny);
                      gy[k] = u[k+nx]- u[k];
                   }
                  }
                j = ny-1;
for(i=0; i< nx; i++) { k = ind_2d1d(i, j, nx, ny);
                         gy[k] = 0;
                }
              }
void projection_chambo(int nx, int ny, float lambda, float u[], float pr[],
                       float tau, float tol, int nmax)
       //      Calcule la projection v d'une image u
      //      sur lambda G, par l'algorithme
     //      proposÈ par Chambolle.
           {
                int  taille=nx*ny,k, n;
                float *px, *py, *w, *gx, *gy;
                float residu, res1, res2, nrm, nrmp;
           px=malloc(taille*sizeof(float));
           py=malloc(taille*sizeof(float));
           gx=malloc(taille*sizeof(float));
           gy=malloc(taille*sizeof(float));
           w=malloc(taille*sizeof(float));
    for (k = 0; k< nx*ny; k++)
              {   px[k]=0;
```

```
                py[k]=0;
                pr[k]= u[k];
            }
            n = 0;
            residu=1.;
            while (residu > tol && n < nmax)
              {   n++;
                  divergence( nx, ny, &px[0], &py[0], &w[0]);
                for (k=0; k< nx*ny; k++)
                    w[k] -= u[k]/lambda;
                 gradientX(nx, ny, w, gx);
                 gradientY(nx, ny, w, gy);
                 nrmp = 0.0;
                 residu = 0.0;
                 for (k=0; k < nx*ny; k++)
               {  nrm  = sqrt( gx[k]*gx[k] +  gy[k]*gy[k] ) ;
                     res1 = tau*(gx[k] - nrm*px[k])/(1 + tau*nrm);
                     res2 = tau*(gy[k] - nrm*py[k])/(1 + tau*nrm);
                   nrmp +=  px[k]*px[k]+ py[k]*py[k];
                     px[k] = (px[k]+tau*gx[k])/(1 + tau*nrm);
                     py[k] = (py[k]+tau*gy[k])/(1 + tau*nrm);
                   residu += res1*res1+ res2*res2;
               }
                   residu = sqrt(residu/nrmp);
                   if(n%10==0)
                   printf("iteration= %d, residu= %6.2e \n",n,residu);
```

```
                  }
                  divergence( nx, ny, &px[0], &py[0], &w[0]);
           for (k=0; k < nx*ny; k++)  pr[k] = u[k]-lambda*w[k];
if (n == nmax)
     printf ("Pas de convergence de l'agorithme de chambolle");
else printf("algo de chambolle convergence au bout de %d iterations \n",n);
          }
float ISNR( int n, float u[], float v[], float w[])
// Si l'ISNR est negatif, la restauration est mauvaise
// Si l'ISNR est positif, alors la restauration est de bonne qualite
// Si l'ISNR est null, alors il n'y a pas eu de restauration.
{
    int i;
    float num, denom, num=0.,denom=0.;
    for (i=0 ; i<n; i++)
    {  num+=(u[i]-v[i])*(u[i]-v[i]);
       denom+=(u[i]-w[i])*(u[i]-w[i]);
    }
    num=num/denom;
    num=10.*log(num)/log(10.);
    return(num);
}
```

```
//*********Semi-quadratique pour R=I *********************
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define max(a,b) (a>=b?a:b)
#define min(a,b) (a<=b?a:b)
void mat_vec(int nx, int ny,  double alpha, double u[],
     double w[],void (*op_blur)(), double (*phipsur)(),
     double beta,double v[], double eps, double lambda,
     double(* noyau)());
void gradient_conjugue(int nx, int ny, double u[] ,
    double tol,double (*phipsur)(), void(*mat_vec)(),
    double w[], double rhs[], double eps, double lambda,
    double(* noyau)(), void(*op_blur)() );
double prod_scal(int size, double u[], double v[]);
void calc_coefb(int size, double gx[], double gy[], double bb[],
         double eps, double (* phipsur)());
void op_blur( int index, int nx, int ny, double u[], double rhs[],
        double (* noyau)(), int supp_x, int supp_y);
int ind_2d1d( int i,  int j, int nx, int ny);
void ind_1d2d( int ind, int i, int j, int nx, int ny);
double noyau_delta( int i, int j);
double noyau_disque( int i, int j);
void semi_quadratic( int nx, int ny, double tol, double retol,
    double(*phipsur)(), void(*mat_vec)(), double u0[], double sm[],
    double eps, double lambda,double(* noyau)(), void(*op_blur));
```

```
double phipsur(double x);
double ISNR( int n, double u[], double v[], double w[]);
double norm2( int size, double u[]);
        main()  {
     int nx, ny, taille, k, itmax =50000;
     int supp_x=10, supp_y=10;
     double lambda = 0.121, eps = 0.01, tol = 1e-4, retol=1e-4;
     double *u, *u0, *sm, *rhs, *w, nrm;
     double (*noyau)();
        FILE *origsm;
        origsm = fopen("origsm.txt","r");
        fscanf(origsm,"%d\n",&nx);
        fscanf(origsm,"%d\n",&ny);
         printf("%d, %d\n", nx, ny);
         taille = nx*ny;
         u  =(double *) malloc(taille*sizeof(double));
         u0 =(double *) malloc(taille*sizeof(double));
         sm =(double *) malloc(taille*sizeof(double));
           for (k = 0; k< nx*ny; k++)
               fscanf(origsm, "%lf\t", &u0[k]);
               fclose(origsm);
           for (k = 0; k< nx*ny; k++)
               fprintf(origsm,"%6.2f\t", u[k]);
               fclose(origsm);
            origsm = fopen("bruitsm.txt", "r");
            for (k = 0; k< nx*ny; k++)
```

```
                fscanf(origsm, "%f\t", &u0[k]);
                fclose(origsm);
semi_quadratic( nx, ny, tol, retol, phipsur, mat_vec, u0,
             sm, eps, lambda, noyau_disque, op_blur );
         printf("ISNR = %6.2e \n", ISNR(nx*ny, u, u0, sm));
                  FILE *oragsm;
                  oragsm = fopen("oragsm.txt","w");
              for (k = 0; k< nx*ny; k++)
                   fprintf(oragsm,"%6.2f\t", sm[k]);
                   fclose(oragsm);
              return (0);  }
 void gradientX(int nx, int ny, double u[], double gx[])
            {
             int i, j, k;
             for(i=0; i< nx-1; i++)
             {
                 for(j=0; j< ny; j++)
                 {  k = ind_2d1d(i, j, nx, ny);
                    gx[k] = u[k+1]- u[k];
                 }  }
              i = nx-1;
              for(j=0; j< ny; j++)
              {   k = ind_2d1d(i, j, nx, ny);
                  gx[k] = 0;
              }  }
 void gradientY(int nx, int ny, double u[], double gy[])
```

```
              {
               int i, j, k;
               for(j=0; j< ny-1; j++)
                 {
                  for(i=0; i< nx; i++)
                  {  k = ind_2d1d(i, j, nx, ny);
                     gy[k] = u[k+nx]- u[k];
                  }  }
               j = ny-1;
               for(i=0; i< nx; i++)
               {  k = ind_2d1d(i, j, nx, ny);
                  gy[k] = 0;
               }  }
void mat_vec( int nx, int ny, double alpha, double u[], double w[],
     void(*op_blur)(),double(*phipsur)(), double beta, double v[],
      double eps, double lambda, double(* noyau)())
       {
          int     i, j, k, taille ;
          double  *bb, *gx, *gy, *zz, *res;
          double  supp_x = 10, supp_y=10;
          taille=nx*ny;
       bb =(double *) malloc(taille*sizeof(double));
       gx =(double *) malloc(taille*sizeof(double));
       gy =(double *) malloc(taille*sizeof(double));
       zz =(double *) malloc(taille*sizeof(double));
       res =(double *) malloc(taille*sizeof(double));
```

```
                gradientX(nx, ny, w, gx);
                gradientY(nx, ny, w, gy);
                calc_coefb(nx*ny, gx, gy, bb, eps, phipsur);
                gradientX(nx, ny, u, gx);
                gradientY(nx, ny, u, gy);
                for (k =0; k < nx*ny; k++)
                {
                 zz[k]=u[k];
                 res[k] = 0.0;
                    j=k/nx;
                    i=k-j*nx;
        if (i < nx)  res[k] =res[k]-  bb[k]*gx[k];
        if (i > 0)   res[k] =res[k]+ bb[k-1]*gx[k-1];
        if (j < ny)  res[k] =res[k]-  bb[k]*gy[k];
        if (j > 0)   res[k] =res[k]+ bb[k-nx]*gy[k-nx];
                     res[k] = lambda*res[k] + zz[k];
                     v[k] = alpha*res[k] + beta*v[k];
                }
                 free(bb);free(gx);free(gy); free(res);free(zz);
             }
    void calc_coefb( int size, double gx[], double gy[], double bb[],
                    double eps, double(* phipsur)())
                {
                 int k;
                 double nrm;
        for (k =0; k < size; k++)
```

```
                {  nrm = sqrt(gx[k]*gx[k] + gy[k]*gy[k]);
         if (nrm <eps)  bb[k] = 0.5*phipsur(eps);
           else if (nrm > 1./eps) bb[k] = 0.5*phipsur(1.0/eps );
           else     bb[k] = 0.5*phipsur(nrm);
                 }  }
    double prod_scal( int size, double u[], double v[])
         {
            double ps;
            int i;
              ps = 0.0;
            for (i=0; i< size; i++)  ps += u[i] * v[i];
             return (ps);
         }
    double norm2( int size, double u[])
         {
            double vnrm;
            int i;
            vnrm = 0.0;
            for (i=0; i< size; i++)
             vnrm += u[i] * u[i];
             vnrm  = sqrt(vnrm);
            return (vnrm);
         }
    double norm_diff( int size, double u[], double v[])
         {
            double vnrm;
```

```
        int i;
        vnrm = 0.0;
        for (i=0; i< size; i++)
             vnrm += (u[i] - v[i])*(u[i] - v[i]);
             vnrm = sqrt(vnrm);
        return (vnrm);
     }
int ind_2d1d( int i, int j, int nx, int ny)
          {
            int ind ;
            ind = j*nx + i;
            return (ind) ;
          }
void ind_1d2d( int ind, int i, int j, int nx, int ny)
          {
            j = ind/nx;
            i = ind - j*nx;
          }
double phipsur(double x)
          {
              double phips;
              phips = 2.0/(sqrt(1.0+ x*x));
              return(phips);
           }
void gradient_conjugue( int nx, int ny, double u[] ,double tol,
      double (*phipsur)(), void (*mat_vec)(), double w[],
```

```
        double rhs[],double eps,double lambda,
        double(* noyau)(), void (*op_blur)())
             {
             int     ni, k,taille;
             double *rr,*dd,*ww;
             double res1, res2, alpha, beta, nrm, nrmww, nrmdd;
             taille = nx*ny;
             rr = (double *) malloc(taille*sizeof(double));
             dd = (double *) malloc(taille*sizeof(double));
             ww = (double *) malloc(taille*sizeof(double));
              for (k =0; k < nx*ny; k++)
                   rr[k]= rhs[k];
                   nrm = norm2(nx*ny, rr);
mat_vec(nx, ny, -1.0, u, w, op_blur, phipsur ,1.0, rr, eps, lambda, noyau);
              for (k =0; k < nx*ny; k++)   ww[k]= -rr[k];
                       nrmww = norm2(nx*ny, ww);
mat_vec(nx, ny,1.0, ww, w, op_blur, phipsur,0.0, dd, eps,lambda, noyau);
                  nrmdd = norm2(nx*ny, dd);
                  res1 = prod_scal(nx*ny, rr, ww);
                  res2 = prod_scal(nx*ny, ww, dd);
                  alpha = res1/res2;
             for (k =0; k < nx*ny; k++) u[k] = u[k] + alpha*ww[k];
                beta = 0.0;
                 for (ni = 1; ni <= nx*ny; ni++)
                 {
             for (k =0; k < nx*ny; k++)  rr[k] = rr[k] - alpha*dd[k];
```

```
                        nrm = norm2(nx*ny, rr)/norm2(nx*ny, u);
                        if( nrm < tol ) break;
                        res1 = prod_scal(nx*ny, rr, dd);
                        res2 = prod_scal(nx*ny, ww, dd);
                        beta = res1/res2;
                for (k =0; k < nx*ny; k++)  ww[k] = -rr[k] + beta*ww[k];
mat_vec(nx, ny,1.0, ww, w, op_blur, phipsur,0.0, dd, eps, lambda, noyau);
                        res1 = prod_scal(nx*ny, rr, ww);
                        res2 = prod_scal(nx*ny, ww, dd);
                        alpha = res1/res2;
                        for (k =0; k < nx*ny; k++) u[k] = u[k] + alpha*ww[k];
                  }
                         nrm = norm2(nx*ny, u);
                         free(rr);free(dd);free(ww);
        }
void semi_quadratic ( int nx, int ny, double tol, double retol,
        double (*phipsur)(),void(*mat_vec)(),double u0[],
        double w[],double eps, double lambda,
        double(* noyau)(), void (*op_blur))
        {
         int         nit, k, itmax, taille;
         double      residu, nrm;
         double      *zz;
            taille=nx*ny;
            zz =(double *) malloc(taille*sizeof(double));
          nit    = 0;
```

```
          retol  = 1.e-4;
          residu = 1.e+12;
          for (k=0; k< nx*ny; k++)
                zz[k] = u0[k];
          while ((nit <= itmax) && (residu >= retol))
           {
               for (k=0; k< nx*ny; k++)
                     w[k] = zz[k];
                nit++;
                nrm = norm2(nx*ny, w);
gradient_conjugue(nx, ny, zz, tol, phipsur, mat_vec, w, u0,
                 eps, lambda, noyau, op_blur);
                residu = norm_diff(nx*ny, w, zz);
                residu = residu / nrm ;
           }
                free(zz);
        }
 double noyau_delta( int i, int j)
             {
               double res;
                res = 0.0;
                if ((i == 0) && (j == 0)) res = 1.0;
                return(res);
             }
 double noyau_disque(int i, int j)
             {
```

```
                  double          res, ray = 10;
                  static int          summ = 0;
                  int             rmax, i1, j1;
                  if (summ == 0)
                  {
                    rmax = ray;
                    for (i1 = -rmax; i1 <=  rmax; i1++)
                      for (j1 =-rmax; j1 <= rmax; j1++)
                      {    if (i1*i1 + j1*j1 <= ray*ray)  summ++;
                      }  }
                   res = 0.;
                   if (i*i + j*j <= ray*ray) res = 1.0/summ;
                   return(res);
                }
 void op_blur( int index, int nx, int  ny, double u[], double res[],
             double(* noyau)(), int supp_x, int supp_y)
//          Le support du noyau est suppose compris dans le rectangle
//          [-supp_x, supp_x]*[-supp_y, supp_y]
//          On suppose ici que l'operateur de op_blur est autoadjoint
//          Il envoie donc la meme chose pour index = 0 ou index =1
            {    int i, j, s, t,  k, st;
                 int smin, smax, tmin, tmax;
                 double resu;
                 for (j = 0; j < ny; j++)
                   for (i = 0; i < nx; i++)
                     { smin =  max(-supp_x, i + 1 - nx);
```

```
                        smax =  min(supp_x, i);
                        tmin =  max(-supp_y, j + 1 - ny);
                        tmax =  min(supp_y, j);
                        k = ind_2d1d(i, j, nx, ny);
                        resu =0.0;
                        for (t = -tmin; t <= tmax; t++)
                         for (s = smin; s <= smax; s++)
                         {   st = ind_2d1d(i-s, j-t, nx, ny);
                             resu = resu + noyau(s,t)*u[st];
                         }   }
           }
double ISNR( int n, double u[], double v[], double w[])
 {
   int i;
   double num, denom;
   num =0.,denom =0.;
   for (i =0 ; i< n; i++)
      {  num += (u[i]-v[i])*(u[i]-v[i]);
         denom += (u[i]-w[i])*(u[i]-w[i]);
      }
   num = num/denom;
   num = 10.*log(num)/log(10.);
   return(num);
 }
```

```
//*************Semi-quadratique (deconvolution)*********************
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define max(a,b) (a>=b?a:b)
#define min(a,b) (a<=b?a:b)
void mat_vec(int nx, int ny,  double alpha, double u[], double w[],
             void (*op_blur)(),double (*phipsur)(), double beta,
             double v[], double eps, double lambda, double(* noyau)());
void gradient_conjugue(int nx, int ny, double u[] , double tol,
    double (*phipsur)(), void(*mat_vec)(), double w[], double rhs[],
    double eps, double lambda, double(* noyau)(), void(*op_blur)() );
double prod_scal(int size, double u[], double v[]);
void calc_coefb(int size, double gx[], double gy[], double bb[],
             double eps, double (* phipsur)());
void op_blur( int index, int nx, int ny, double u[], double rhs[],
             double (* noyau)(), int supp_x, int supp_y);
int ind_2d1d( int i,  int j, int nx, int ny);
void ind_1d2d( int ind, int *i, int *j, int nx, int ny);
double noyau_delta( int i, int j);
double noyau_disque( int i, int j);
void semi_quadratic( int nx, int ny, double tol, double retol,
    double(*phipsur)(), void(*mat_vec)(), double u0[], double sm[],
    double eps, double lambda,double(* noyau)(), void(*op_blur)());
double phipsur(double x);
double ISNR( int n, double u[], double v[], double w[]);
```

```
double SNR(int n,double *u,double *v);
double norm2( int size, double u[]);
        main()
           {
     int nx, ny, taille, k, itmax =50000;
     int supp_x=10, supp_y=10;
     double lambda = 0.001, eps = 0.01, tol = 1e-8, retol=1e-8;
     double *u, *u0, *sm, *rhs, *w, nrmm;
     double (*noyau)();
        FILE *origsm;
        origsm = fopen("origsm.txt","r");
        fscanf(origsm,"%d\n",&nx);
        fscanf(origsm,"%d\n",&ny);
         printf("%d, %d\n", nx, ny);
         taille = nx*ny;
    u  =(double *) malloc(taille*sizeof(double));
    u0 =(double *) malloc(taille*sizeof(double));
    sm =(double *) malloc(taille*sizeof(double));
// FLOUAGE ET EVENTUELLEMENT BRUITAGE
          for (k = 0; k< nx*ny; k++)
               fscanf(origsm, "%lf\t", &u[k]);
               fclose(origsm);
         op_blur(0, nx, ny, u, u0, noyau_disque, supp_x, supp_y);
              FILE *orgsm;
                  orgsm = fopen("orgsm.txt","w");
              for (k = 0; k< nx*ny; k++)
```

```
                        fprintf(orgsm,"%6.2f\t", u0[k]);
                        fclose(orgsm);
semi_quadratic( nx, ny, tol, retol, phipsur, mat_vec, u0, sm, eps, lambda,
               noyau_disque, op_blur );
         printf("ISNR = %6.2f \n", ISNR(nx*ny, u, u0, sm));
         printf ("SNR = %6.2f \n", SNR( nx*ny, u0, sm));
              FILE *oragsm;
              oragsm = fopen("oragsm.txt","w");
              for (k = 0; k< nx*ny; k++)
                   fprintf(oragsm,"%6.2f\t", sm[k]);
                   fclose(oragsm);
              return (0);
               }
 void gradientX(int nx, int ny, double u[], double gx[])
           {
           int i, j, k;
            for(i=0; i< nx-1; i++)
            {
                for(j=0; j< ny; j++)
                { k = ind_2d1d(i, j, nx, ny);
                  gx[k] = u[k+1]- u[k];
                }
            }
             i = nx-1;
             for(j=0; j< ny; j++)
             {   k = ind_2d1d(i, j, nx, ny);
```

```
                    gx[k] = 0;
                }
              }
void gradientY(int nx, int ny, double u[], double gy[])
               {
                int i, j, k;
                for(j=0; j< ny-1; j++)
                  {
                   for(i=0; i< nx; i++)
                   {  k = ind_2d1d(i, j, nx, ny);
                      gy[k] = u[k+nx]- u[k];
                   }
                  }
                j = ny-1;
                for(i=0; i< nx; i++)
                {  k = ind_2d1d(i, j, nx, ny);
                   gy[k] = 0;
                }
               }
 void mat_vec( int nx, int ny, double alpha, double u[],
   double w[], void(*op_blur)(),double(*phipsur)(),
   double beta, double v[], double eps, double lambda, double(* noyau)())
        {
             int     i, j, k, taille ;
             double  *bb, *gx, *gy, *zz, reso;
             int  supp_x = 10, supp_y=10;
```

```
            taille=nx*ny;
           bb =(double *) malloc(taille*sizeof(double));
           gx =(double *) malloc(taille*sizeof(double));
           gy =(double *) malloc(taille*sizeof(double));
           zz =(double *) malloc(taille*sizeof(double));
              op_blur(0, nx, ny, u, gx, noyau, supp_x, supp_y);
              op_blur(1, nx, ny, gx, zz, noyau, supp_x, supp_y);
           gradientX(nx, ny, w, gx);
           gradientY(nx, ny, w, gy);
              calc_coefb(nx*ny, gx, gy, bb, eps, phipsur);
           gradientX(nx, ny, u, gx);
           gradientY(nx, ny, u, gy);
            for (k =0; k < nx*ny; k++)
            {   reso = 0.0;
                j=k/nx;
                i=k-j*nx;
               if (i < nx)  reso -=  bb[k]*gx[k];
               if (i > 0)   reso +=  bb[k-1]*gx[k-1];
               if (j < ny)  reso -=  bb[k]*gy[k];
               if (j > 0)   reso += bb[k-nx]*gy[k-nx];
                            reso = lambda*reso + zz[k];
              v[k] = alpha*reso + beta*v[k];
            }
             free(bb);free(gx);free(gy);free(zz);
         }
void calc_coefb( int size, double gx[], double gy[], double bb[],
```

```
                      double eps, double(* phipsur)())
                 {
                  int k;
                  double nrm;
                for (k =0; k < size; k++)
                  {
                    nrm = sqrt(gx[k]*gx[k] + gy[k]*gy[k]);
                    if (nrm <eps)  bb[k] = 0.5*phipsur(eps);
                    else if (nrm > 1./eps)
                       bb[k] = 0.5*phipsur(1.0/eps );
                    else
                       bb[k] = 0.5*phipsur(nrm);
                   }
                 }
    double prod_scal( int size, double u[], double v[])
         {
            double ps;
            int i;
            ps = 0.0;
            for (i=0; i< size; i++) ps += u[i] * v[i];
             return (ps);
         }
    double norm2( int size, double u[])
         {
            double vnrm;
            int i;
```

```
        vnrm = 0.0;
        for (i=0; i< size; i++)
             vnrm += u[i] * u[i];
             vnrm  = sqrt(vnrm);
        return (vnrm);
     }
double norm_diff( int size, double u[], double v[])
     {
        double vnrm;
        int i;
        vnrm = 0.0;
        for (i=0; i< size; i++)
             vnrm += (u[i] - v[i])*(u[i] - v[i]);
             vnrm = sqrt(vnrm);
        return (vnrm);
     }
int ind_2d1d( int i, int j, int nx, int ny)
          {
            int ind ;
            ind = j*nx + i;
            return (ind) ;
          }
void ind_1d2d( int ind, int *i, int *j, int nx, int ny)
          {
            *j = ind/nx;
            *i = ind - (*j)*nx;
```

```
            }
 double phipsur(double x)
            {
               double phips;
                phips = 2.0/(sqrt(1.0+ x*x));
               return(phips);
             }
 void gradient_conjugue( int nx, int ny, double u[] ,double tol,
               double (*phipsur)(), void (*mat_vec)(), double w[],
                double rhs[],double eps, double lambda, double(* noyau)(),
                 void (*op_blur)())
             {
             int     ni, k,taille;
             double  *rr, *dd,*ww;
             double res1, res2, alpha, beta, nrm, nrmww, nrmdd;
             taille = nx*ny;
             rr = (double *) malloc(taille*sizeof(double));
             dd = (double *) malloc(taille*sizeof(double));
             ww = (double *) malloc(taille*sizeof(double));
              for (k =0; k < nx*ny; k++) rr[k]= rhs[k];
mat_vec(nx, ny, -1.0, u, w, op_blur, phipsur ,1.0, rr, eps, lambda, noyau);
              for (k =0; k < nx*ny; k++) ww[k]= -rr[k];
              nrmww = norm2(nx*ny, ww);
mat_vec(nx, ny, 1.0, ww, w, op_blur, phipsur, 0.0, dd, eps,lambda, noyau);
              res1 = prod_scal(nx*ny, rr, ww);
              res2 = prod_scal(nx*ny, ww, dd);
```

```
                alpha = res1/res2;
                for (k =0; k < nx*ny; k++) u[k] = u[k] + alpha*ww[k];
                beta = 0.0;
                for (ni = 1; ni <= nx*ny; ni++)
                   {
                   for (k =0; k < nx*ny; k++)  rr[k] = rr[k] - alpha*dd[k];
                       nrm = norm2(nx*ny, rr)/norm2(nx*ny, u);
                       if( nrm < tol ) break;
                          res1 = prod_scal(nx*ny, rr, dd);
                          res2 = prod_scal(nx*ny, ww, dd);
                          beta = res1/res2;
                   for (k =0; k < nx*ny; k++)  ww[k] = -rr[k] + beta*ww[k];
mat_vec(nx, ny, 1.0, ww, w, op_blur, phipsur, 0.0, dd, eps, lambda, noyau);
                       res1 = prod_scal(nx*ny, rr, ww);
                       res2 = prod_scal(nx*ny, ww, dd);
                       alpha = res1/res2;
                for (k =0; k < nx*ny; k++) u[k] = u[k] + alpha*ww[k];
                  }
                        nrm = norm2(nx*ny, u);
                        free(rr);free(dd);free(ww);
         }
void semi_quadratic ( int nx, int ny, double tol, double retol,
         double (*phipsur)(), void(*mat_vec)(), double u0[],
         double w[],double eps, double lambda, double(* noyau)(),
         void (*op_blur)())
        {
```

```
          int         nit, k, itmax, taille, supp_x=10, supp_y=10;
          double      residu, nrm;
          double      *zz,*scd;
             taille=nx*ny;
             zz  =(double *) malloc(taille*sizeof(double));
             scd =(double *) malloc(taille*sizeof(double));
           nit    = 0;
           retol  = 1.e-8;
           residu = 1.e+12;
           for (k=0; k< nx*ny; k++)
                 zz[k] = u0[k];
           while ((nit <= itmax) && (residu >= retol))
            {
                for (k=0; k< nx*ny; k++) w[k] = zz[k];
                 nit++;
                 nrm = norm2(nx*ny, w);
op_blur(0, nx, ny, u0, scd, noyau_disque, supp_x, supp_y);
gradient_conjugue(nx, ny, zz, tol, phipsur, mat_vec, w, scd,
                 eps, lambda, noyau, op_blur);
                 residu = norm_diff(nx*ny, w, zz);
                 residu = residu / nrm ;
            }
                 free(zz);
        }
 double noyau_delta( int i, int j)
              {
```

```
                  double res;
                  res = 0.0;
                  if ((i == 0) && (j == 0)) res = 1.0;
                  return(res);
               }
 double noyau_disque(int i, int j)
               {
                 double                         res;
                 static int       summ = 0, ray = 2;
                 int                   rmax, i1, j1;
                 if (summ == 0)
                 { rmax = ray;
                   for (i1 = -rmax; i1 <=  rmax; i1++)
                     for (j1 =-rmax; j1 <= rmax; j1++)
                     {   if (i1*i1 + j1*j1 <= ray*ray)  summ++;
                     }
                  }
                  res = 0.;
                  if (i*i + j*j <= ray*ray) res = 1.0/summ;
                  return(res);
               }
 void op_blur( int index, int nx, int  ny, double u[], double res[],
                double(* noyau)(),int supp_x, int supp_y)
//          Le support du noyau est suppose compris dans le rectangle
//          [-supp_x, supp_x]*[-supp_y, supp_y]
//          On suppose ici que l'operateur de op_blur est autoadjoint
```

```
//          Il envoie donc la meme chose pour index = 0 ou index =1
            {
                 int i, j, s, t,  k, st;
                 int smin, smax, tmin, tmax;
                 double resu;
                 for (j = 0; j < ny; j++)
                   for (i = 0; i < nx; i++)
                     {   smin =  max(-supp_x, i + 1 - nx);
                         smax =  min(supp_x, i);
                         tmin =  max(-supp_y, j + 1 - ny);
                         tmax =  min(supp_y, j);
                         k = ind_2d1d(i, j, nx, ny);
                         resu = 0.0;
                       for (t = tmin; t <= tmax; t++)
                        for (s = smin; s <= smax; s++)
                        {   st = ind_2d1d(i-s, j-t, nx, ny);
                            resu = resu + noyau(s,t)*u[st];
                        }
                        res[k] = resu;
                   } }
 double ISNR( int n, double u[], double v[], double w[])
// Si l'ISNR est negatif, la restauration est mauvaise
// Si l'ISNR est positif, alors la restauration est de bonne qualite
// Si l'ISNR est null, alors il n'y a pas eu de restauration.
  {
    int i;
```

```
    double num, denom;
    num =0.,denom =0.;
    for (i =0 ; i< n; i++)
       {  num += (u[i]-v[i])*(u[i]-v[i]);
          denom += (u[i]-w[i])*(u[i]-w[i]);
       }
    num = num / denom;
    num = 10.*log(num)/log(10.);
    return(num);
  }
  double SNR(int n,double *u,double *v)
{
    double num, denom;
    int i;
    num =0.,denom =0.;
    for (i =0 ; i < n; i++)
        {  denom += (u[i]-v[i])*(u[i]-v[i]);
           num += u[i]*u[i];
        }
    num = num / denom;
    num = 10.*log(num)/log(10);
    return(num);
}
```

Table des figures

Bibliographie

[1] R. Adams. *Sobolev Spaces*. Pure and applied Mathematics. Academic Press, Inc, 1975.

[2] L. Alvarez, F. Guichard, P.-L. Lions, and J.-M. Morel, *Axioms and fundamental equations of image processing*. Arch. Rational Mech. Anal. 123 (1993), no. 3, 199-257.

[3] L. Ambrosio, N. Fusco, et D. Pallara. *Functions of bounded variations and free discontinuity problems*. Oxford mathematical monographs. Oxford University Press, 2000.

[4] G. Aubert et P. Kornprobst. *Mathematical problems in image processing. Partial differential equations and the calculs of variations*. volume 147 of Applied Mathematical Science. Springer-Verlag, 2002.

[5] G. Aubert, P. Kornprobst et R. Deriche. *Nonlinear operators in image restoration*. Masson, 1990.

[6] G. Aubert, M. Barlaud, L. Blanc-Feraud et P. Charbonnier. *Deterministic edge-preserving regularization in computed imaging*. IEEE Trans. Imag. Process, 6(2), Feb. 1997.

[7] J.-F. Aujol. *Traitement d'images par approches variationnelles et équations aux dérivées partielles*. 16 avril 2005.

[8] M.Bergounioux. *Quelques méthodes mathématiques pour le traitement d'image*. 30 mai 2006.

[9] D. P. Bertsekas ,A. Nedié et A. E. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, belmont, Massachusetts. USA, 2003.

[10] H. Brézis. *Analyse fonctionnelle*. Théorie et applications. Mathématiques appliquées pour la maitrise. Masson, 1983.

[11] F. Catté, P. L. Lions, J. M. Morel, et T. Coll. *Image selective smoothing and edge detection by nonlinear diffusion*. SIAM J.Numer, 29 :181-193,1992.

[12] P. G. Ciarlet. *Introduction a l'analyse numérique matricielle et á l'optimisation*. Masson, Paris, 1990.

[13] A. Chambolle, et P. L. Lions, *Image Recovery via Total Variation Minimization and Related Problems*, Numer. Math, Vol. 76, 2(1997) ; pp. 167-188.

[14] A. Chambolle, *An algorithm for totale minimization and applications*, Numer. Math,(2004).

[15] T. Chan, et J. Shen. *Image Processing and Analysis Variational, PDE, Wavelet and Stochastic Methods*. Applied Mathematical Science. Siam, 2005.

[16] P. Charbonnier, G. Aubert, L. Blanc-Feraud et M. Barlaud. *Two determinist half-quadratic regularization algorithms for computed*

imaging. Firts IEEE Internat. conf. on Image Processing, Vol. II,Austin, TX, IEEE, Pistaway, NJ, 1994, pp. 168-172.

[17] P. Charbonnier, G. Aubert, L. Blanc-Feraud et M. Barlaud. *Determinist edge-preserving regularization in computed imaging.* IEEE trans. Image Processing, 6(1997), pp. 298-311.

[18] R. Deriche et O. Faugeras. *Les EDP en Traitement des Images et Vision par Ordinateur*. Traitement du Signal, 13(6), 1996. Numéro spécial RFIA 96.

[19] D. Azé. *Eléments d'analyse convexe et variationnelle*, édition marketing S.A, ellipes, 1997.

[20] I. Ekeland et R. Temam. *Analyse convexe et problèmes variationnels*, volume 224 of Grundlehren der mathematischen Wissenschaften. Dunod, second edition, 1983.

[21] L. C. *Evans. Partial Differential Equations*, volume 19 of Graduate Studies in Mathematics. American Mathematical Society, 1991.

[22] L. C. Evans and R. F. Gariepy. *Measure theory and fine properties of functions*, volume 19 of Studies in Advanced Mathematics. CRC Press, 1992.

[23] H. Florent. *Introduction au traitement numérique des images*. 27 mai 2007.

[24] H. Florent. *Techniques de débruitage d'images*. 23 Janvier 2008.

[25] N. Fortier, G. Demoment et Y. Goussard. *GCV and ML methods of determining parameters in image restoration by regularization : Fast computation in the satial domaine and experi-*

mental comparaison. Journal of Visual Comunication and Image Representation, 4(1), March 1993.

[26] D. Geman et G. Reynolds. *Constrained restoration and the recovery of discontinuities*. IEEE Transactions on Pattern Analysis And Machin intelligence, 14(3) : 367-383. March 1992.

[27] D. Gilbarg et N. S. Trudinger.*Elliptic Partial Differential Equations of Second Order*, volume 28 of Princeton Mathematical Series. Springer-Verlag, 1970.

[28] E. Giusti. *Minimal surfaces and functions of bounded variation.* Birkhauser, 1994.

[29] J. Hadamard. *Lectures on Cauchy's Problem in linear Partial Differential Equations*. Yale University Press. New Haven, 1923.

[30] J. B. Hiriart-Urruty and C. Lemarechal.*Convex Analysis ans Minimisation Algorithms I*, volume 305 of Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1993.

[31] V. E. Johson, W. E. Wong, X. Hu et J. P. Hugonin. *Inverse scattering : an iterative numerical method for electronic imaging.* IEEE trans. On Antennas and Propagation, AP-39 : 1742-1751, 1991

[32] J. L. Lions and E. Magenes. *Problèmes aux limites non homogènes*, volume 1. Dunod, 1968.

[33] L. Justen. *Blind Deconvolution. Theory, regularization and applications.*

[34] J. Malik et P.Perona. *A scale space and detection using anisotropique diffusion*. IEEE Comp. Soc. Workshop on comp. Vision, Miami, pages16-22,1987.

[35] M. E. Moghaddam. *Out of Focus Blur Estimation Using Genetic Algorithm*. Journal of Computer Science 4 (4) : 298-304, 2008.

[36] N. Nordström. *Biazed anisotropic diffusion- a unified regularization and diffusion approach to edge detection*. Image Vision Compt ; 8 : 318-327, 1990.

[37] J. Malik et P. Perona. *Scale-space and edge detection using anisotropic diffusion*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(7) :629ñ639, July 1990.

[38] T. Rockafellar. *Convex Analysis*, volume 224 of Grundlehren der mathematischen Wissenschaften. Princeton University Press, second edition, 1983.

[39] L. Rudin, S. Osher, et E. Fatemi. *Nonlinear total variation based noise removal algorithms*. Physica D, 60 :259ñ268, 1992.

[40] A. N. Tikhonov et V. Y. Arsenin. *Solutions of ill-posed problems*. Winston and Wiley, 1977.

[41] A. N. Tikhonov. *Regularization of incorrestly posed problems*. Sov. Math. Dokl, 4 :1624-1627, 1963.