Optimisation pour les méthodes variationnelles appliquées au traitement d'image.

Rapport de stage.

Fabien PIERRE.

TdSI, Master 1.

Université Bordeaux 1.

Objectifs et déroulement du stage. [1]

Durant une fin d'année scolaire au temps maussade, mon stage de deux mois m'a transporté dans un monde radieux bien qu'en niveau de gris. J'ai eu l'occasion de travailler sur des sujets qui attiraient ma curiosité et découvrir des choses que je n'imaginais même pas.

Dans ce rapport, je présenterai les outils de base de la minimisation lisse, avec et sans contrainte. Puis je donnerai des applications en traitement d'image (restauration, inpainting et déconvolution avec les normes L2 et L1-L2 (quasi-Huber)). Ensuite je présenterai des outils de base de l'optimisation non-lisse (Transformée de Fenchel-Legendre, sous-différentielle etc.) Puis je donnerai des applications au traitement d'image, déconvolution, débruitage et inpainting avec des méthodes par minimisation de la variation totale (basée essentiellement sur l'algorithme de Chambolle) ou du critère TV-L1. J'introduirai à cet effet le principe général du Forward-Backward ainsi que des méthodes d'accélération (Beck-Téboulle et l'algorithme primal-dual de Chambolle-Pock). Le stage s'est terminé plus tardivement que prévu sur l'implémentation en Java et C d'un des algorithme les plus avancés du stage.

La démonstration des divers algorithmes implémentés n'a pas forcément été étudiée en détail. Cela a été néanmoins le cas des preuves de convergence des algorithmes de minimisation lisse avec et sans contraintes dans un but de former des développements pour des leçons de l'agrégation (compacité, suites numériques, espaces de Hilbert, etc.) L'accent a été essentiellement mis sur l'implémentation, chose avec laquelle je n'étais pas familier (choix des paramètres, entre autre) et de me donner une idée sur l'optimisation en me fournissant un catalogue de méthodes standards le plus riche et large possible. Je n'ai pas eu le temps, hélas, d'aborder la méthode d'Uzawa et d'étudier avec plus de profondeur les relations de Karush-Kuhn-Tucker dont j'ai tout de même survolé les principes généraux pour me forger une bonne idée dessus. Je pense en outre m'être forgé une idée sérieuse sur l'idée du travail de chercheur car j'ai pu suivre l'équipe de travail "Image" de l'IMB, assister au réunions et séminaires, et être ainsi immergé dans le monde de la recherche universitaire.

Remerciements.

Je tiens à remercier Jean-François Aujol pour sa compétence, son dévouement, et de m'avoir encouragé tout au long de l'année à repasser l'agrégation, un bienveillant conseil qui fut couronné d'un succès. Je ne l'aurai sans doute pas fait sinon : qu'il trouve ici un modeste signe de ma profonde reconnaissance. Je tiens aussi à remercier Romain Yildizoglu pour ses compétences aussi diverses que variées, ainsi que son exemplarité.

Je tiens aussi à remercier les personnes dont j'ai utilisé l'image sans autorisation préalable, et dont les jolis portraits ont égayé mon rapport. Merci donc à Lenna, Loulou de la falaise (posthume), Marisa Berenson ainsi que Michèle Morgan.

1. Résumé d'une demi-page environ et ne dépassant pas une page.

Table des matières

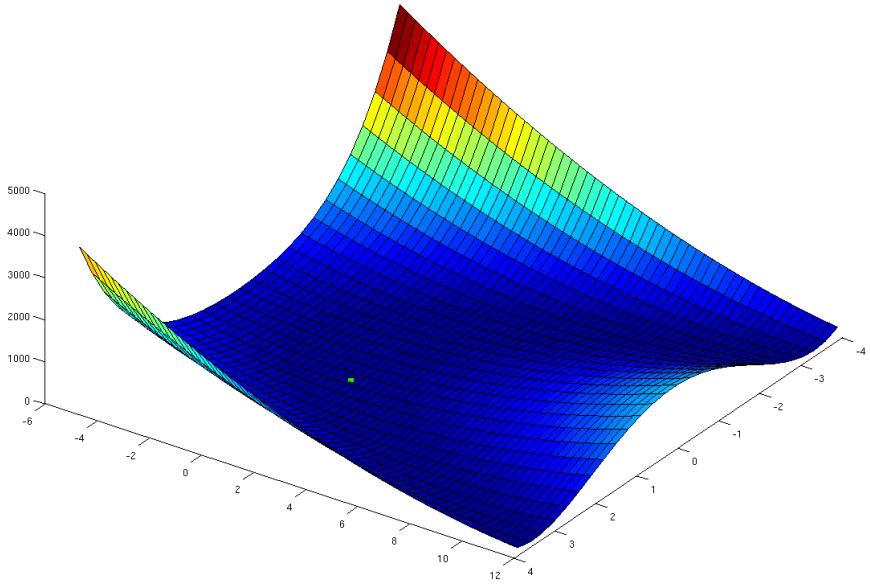
1 Catalogue d'algorithmes de minimisation lisse sans contrainte.

Dans cette section, on cherchera à minimiser la *fonctionnelle*

$$J(x,y) = (x-1)^2 + 10(x^2-y)^2.$$

Ici la fonction est facile à minimiser "à la main" car on peut calculer très facilement son gradient et sa matrice Hessienne. Elle va nous servir de cobaye tout au long de cette première section afin de voir quels sont les avantages et inconvénients des méthodes numériques standards et leurs enjeux. Sur la figure 1 on remarque qu'il y a une vallée en forme de banane (dont la fonction tire son nom de "Banane de Rosenbrock"). Cette vallée va jouer un rôle crucial dans la plupart des méthodes mises en jeux.

FIGURE 1 – Visualisation de la fonctionnelle J.



On calcule pour cela le gradient :

$$\nabla(x,y) = \begin{pmatrix} 2x - 2 + (40(x^2-y))x \\ -20x^2 + 20y \end{pmatrix},$$

et la matrice hessienne :

$$H(x,y) = \begin{pmatrix} 2 + 120x^2 - 40y & -40x \\ -40x & 20 \end{pmatrix}.$$

L'équation d'Euler s'écrit $x^2 = y$ par la deuxième coordonnée du gradient, et la première donne $2x - 2 = 0$, d'où l'on obtient que le seul extrema éventuel est en $(1,1)$. Or

$$H(1,1) = \begin{pmatrix} 82 & -40 \\ -40 & 20 \end{pmatrix},$$

qui est une matrice définie positive, donc J atteint un minimum local en $(1,1)$, qui est un minimum global car la fonctionnelle est coercive.

1.1 Gradient à pas fixe.

Une première méthode utilisée est la méthode du gradient à pas fixe. Il s'agit d'une méthode à point fixe du type $x_{k+1} \leftarrow x_{k-1} - f(x_{k-1})$ utilisée pour trouver le zéro du gradient :

$$x_k \leftarrow x_{k-1} - \nu \nabla J(x_{k-1}),$$

où ν est un paramètre dont dépend l'éventuelle convergence de la méthode ainsi que sa vitesse de convergence s'il y a lieu, et la précision de sa convergence. En effet, le théorème du point fixe de Picard (voir [6] par exemple) nous dit qu'une fonction contractante dans un espace de Banach admet un unique point fixe. le paramètre ν sert précisément à rendre notre fonction de départ contractante si elle ne l'est pas dès le départ. On obtient les itérations suivantes :

FIGURE 2 – Itérations du gradient à pas fixe suivant la "vallée".

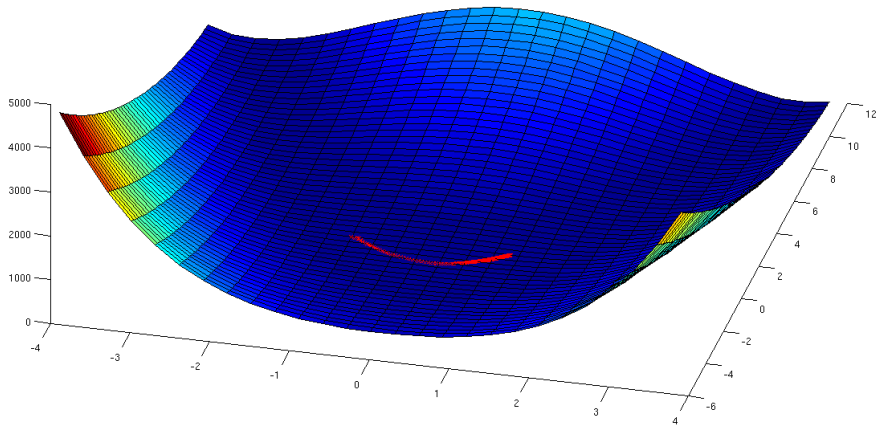
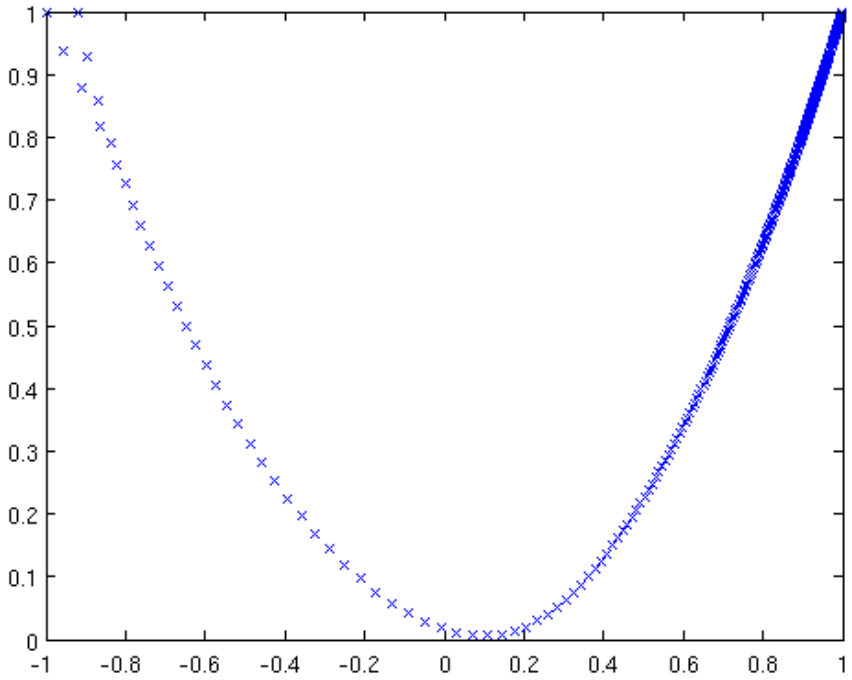


FIGURE 3 – Itérations du gradient à pas fixe.



Cette méthode est relativement rapide car elle demande peu de calcul lors de chaque itération, même s'il faut beaucoup d'itérations pour arriver à une solution fine.

1.2 Gradient à pas optimal.

La méthode du gradient à pas fixe pose un problème cornélien. En effet, si le pas est choisit trop petit, l'algorithme ne converge pas rapidement s'il est trop grand il tourne autour de la solution sans s'en approcher ou bien diverge. Dans la méthode de gradient à pas optimal on ne prend pas un pas fixé initialement mais on cherche à chaque itération le pas qui permettra de diminuer la fonctionnelle le plus possible : à chaque itération :

- résoudre $J(u_k - \rho_k \nabla(u_k)) = \min_{\rho \in \mathbb{R}^+} J(u_k - \rho \nabla(u_k))$
- puis $u_{k+1} \leftarrow u_k - \rho_k \nabla(u_k)$.

On obtient les résultats suivants :

FIGURE 4 – Itérations du gradient à pas optimal suivant la "vallée".

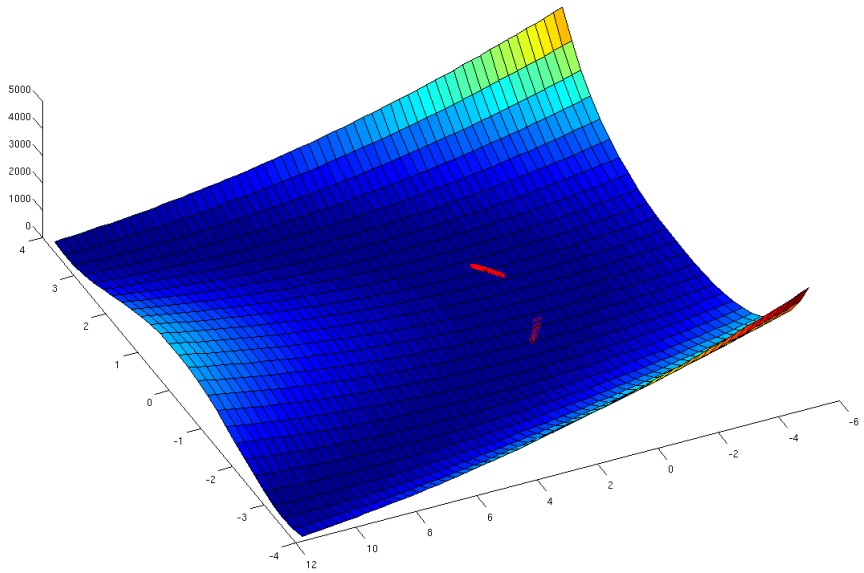
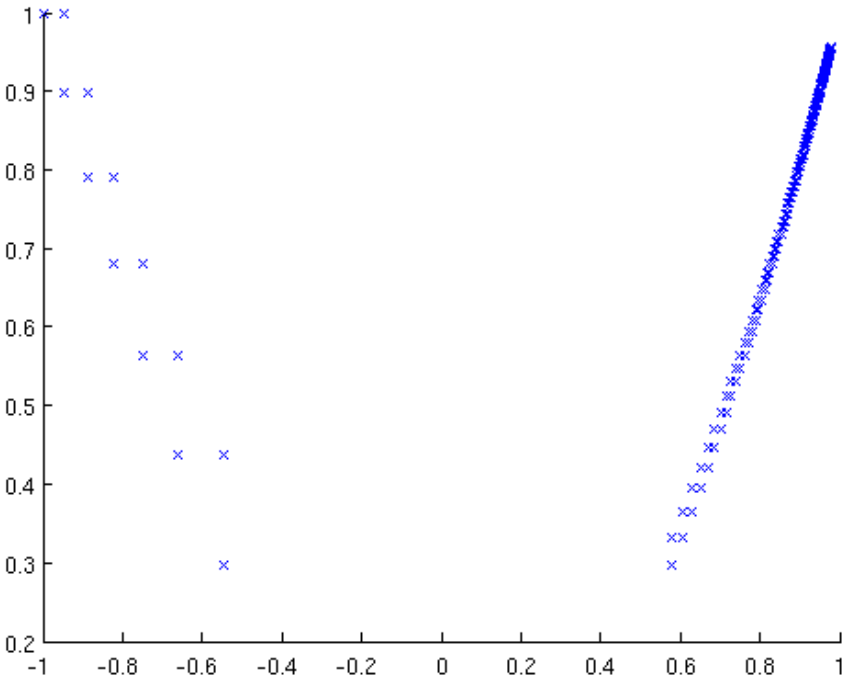


FIGURE 5 – Itérations du gradient à pas optimal optimal.



L'algorithme du gradient à pas optimal ne privilégie pas de direction mais seulement un paramètre, c'est pourquoi il lui faut quand même beaucoup d'itérations. En revanche, dans les zones où la géométrie lui permet de faire un grand pas car il il y a une descente faible mais bien réelle, il descend effectivement de façon optimale.

1.3 Gradient conjugué.

Dans la méthode du gradient conjugué on tient compte de la géométrie de la surface. Ainsi n'on utilise pas directement le gradient optimal tel quel, il est un peu modifié :

$$u_{k+1} \leftarrow u_k - r_k d_k$$

avec

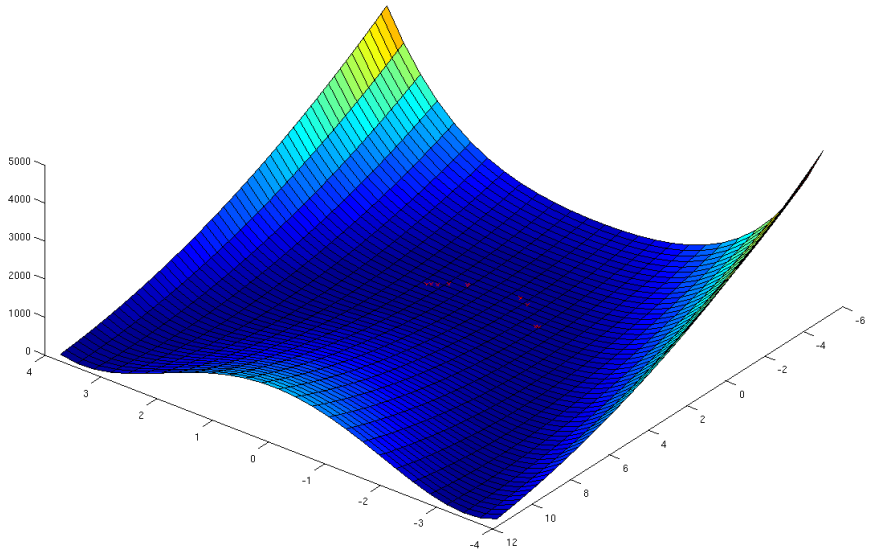
$$J(u_{k+1}) = \inf_{r \in \mathbb{R}} J(u_k - r d_k)$$

et les d_k sont définies par [2] $d_o = \nabla(u_0)$ et

$$d_k \leftarrow \nabla J(u_k) + \frac{(\nabla J(u_k), \nabla J(u_k) - \nabla J(u_{k-1}))}{||\nabla J(u_{k-1})||^2} d_{k-1}.$$

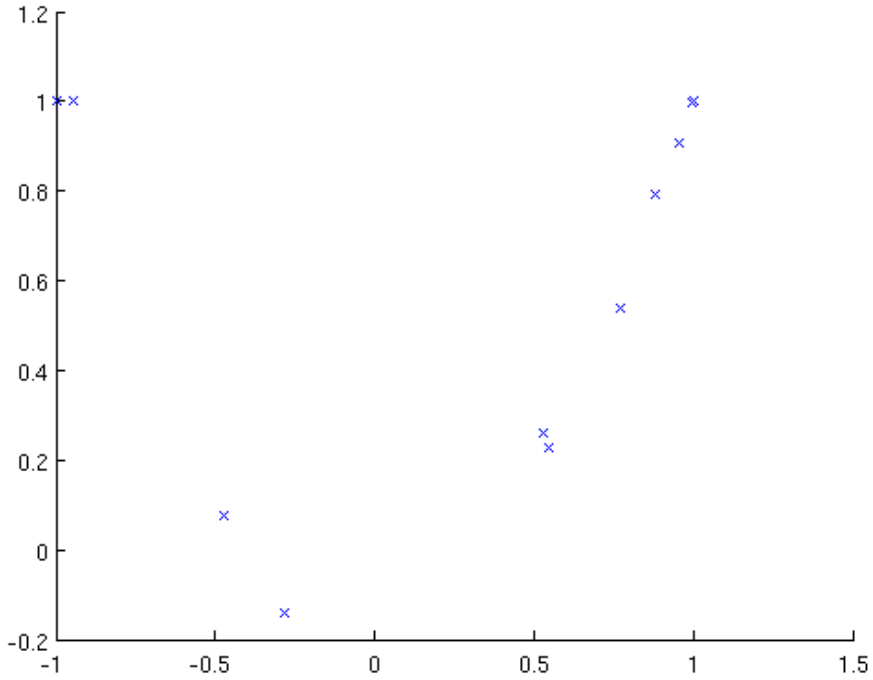
On obtient les résultats suivants :

FIGURE 6 – Itérations du gradient conjugué suivant la "vallée".



2. Il s'agit ici de la version de Polak-Ribière que l'on peut trouver dans [4], mais ce n'est pas la seule qui existe.

FIGURE 7 – Itérations du gradient conjugué.



On remarque que la méthode nécessite moins d'itérations, en revanche le temps de calcul est quand même élevé car le calcul de la direction est non négligeable ainsi que le temps de calcul du paramètre optimal. On remarque aussi que cet algorithme suit encore "la vallée" mais il y avance rapidement.

1.4 Méthode de relaxation.

On minimise coordonnée à coordonnée la fonctionnelle concernée.
On obtient les résultats suivants :

FIGURE 8 – Itérations de la méthode de relaxation sans contraintes suivant la "vallée".

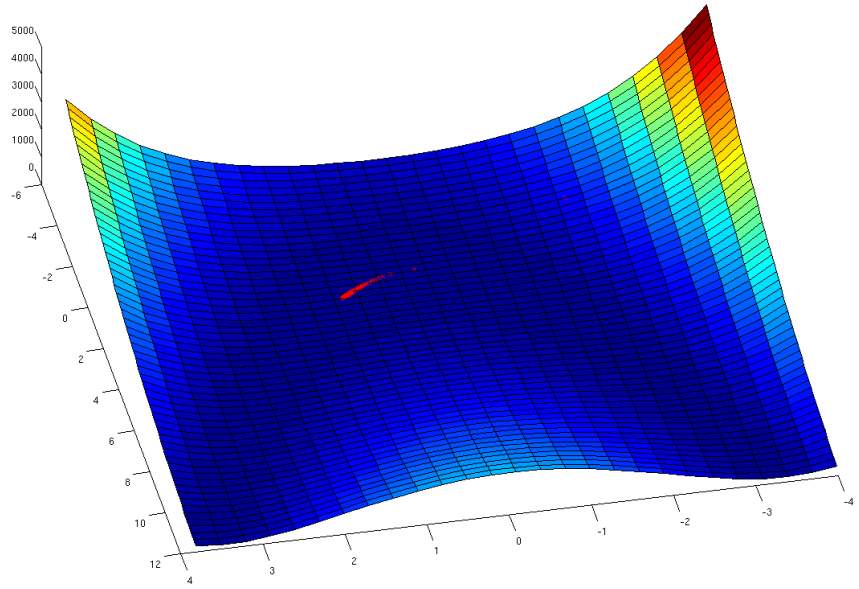
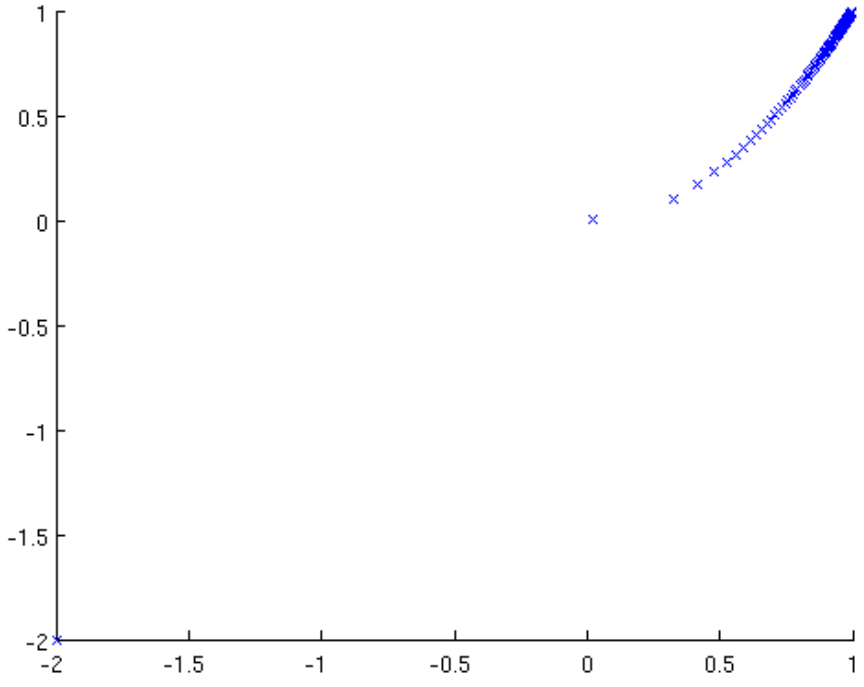


FIGURE 9 – Itérations de la méthode de relaxation sans contraintes.



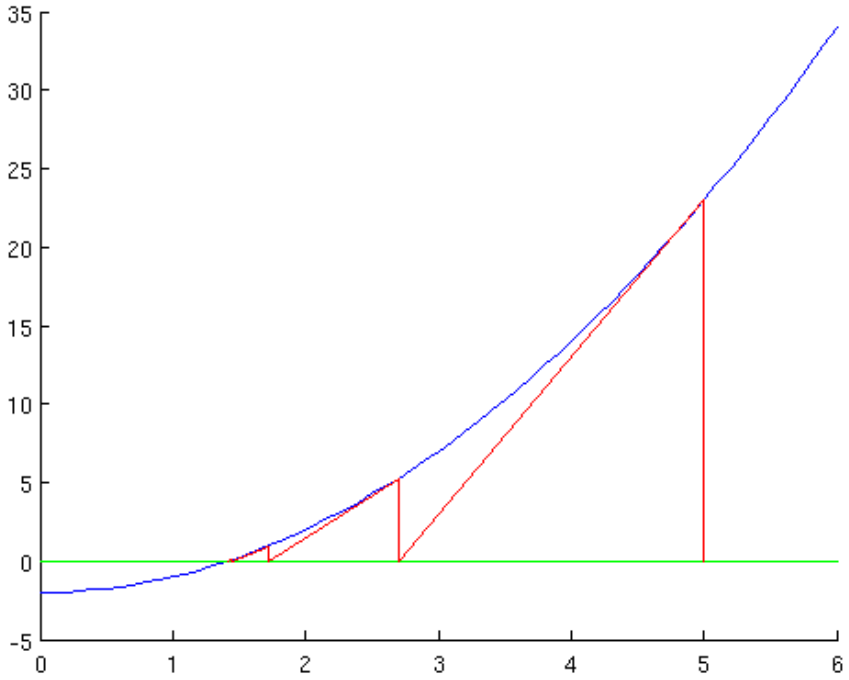
1.5 Méthode de Newton.

La méthode de newton tient compte de la différentielle d'ordre 2 pour optimiser encore plus la méthode de descente de gradient. Dans le cas de la recherche du zéro d'une fonction,

on décide d'approcher une fonction C^1 par sa tangente. i.e. :

$$x_{k+1} \leftarrow x_k - \frac{f(x_k)}{f'(x_k)}.$$

FIGURE 10 – Méthode de Newton utilisée pour approximer $\sqrt{2}$.



Dans le cas de la recherche du minimum où l'on cherche la racine du gradient, l'algorithme se généralise :

$$x_{k+1} \leftarrow x_k - HJ(x_k)^{-1}\nabla J(x_k)$$

On obtient les résultats suivants :

FIGURE 11 – Itérations de la méthode de Newton en 3-d.

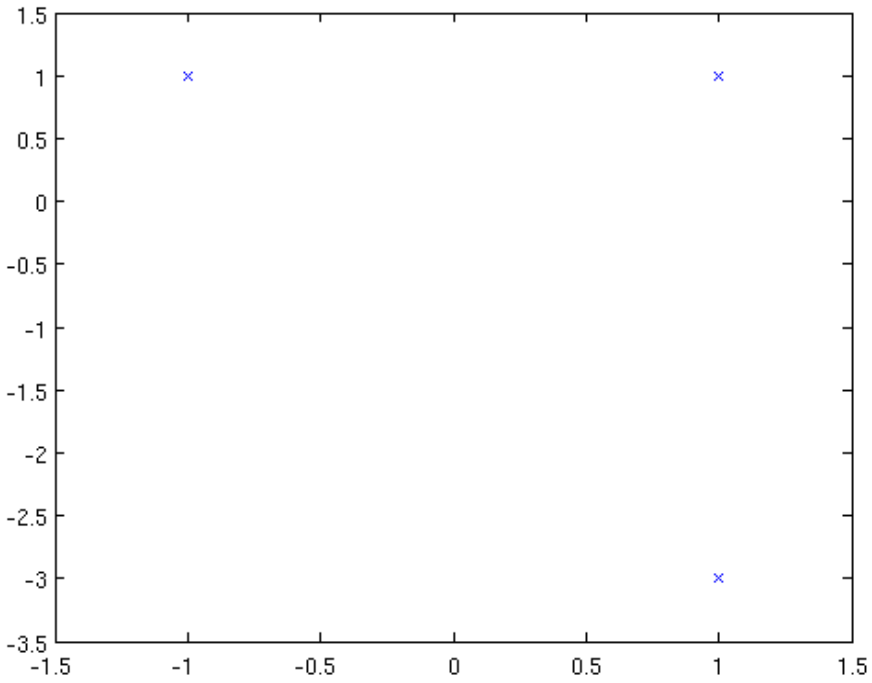
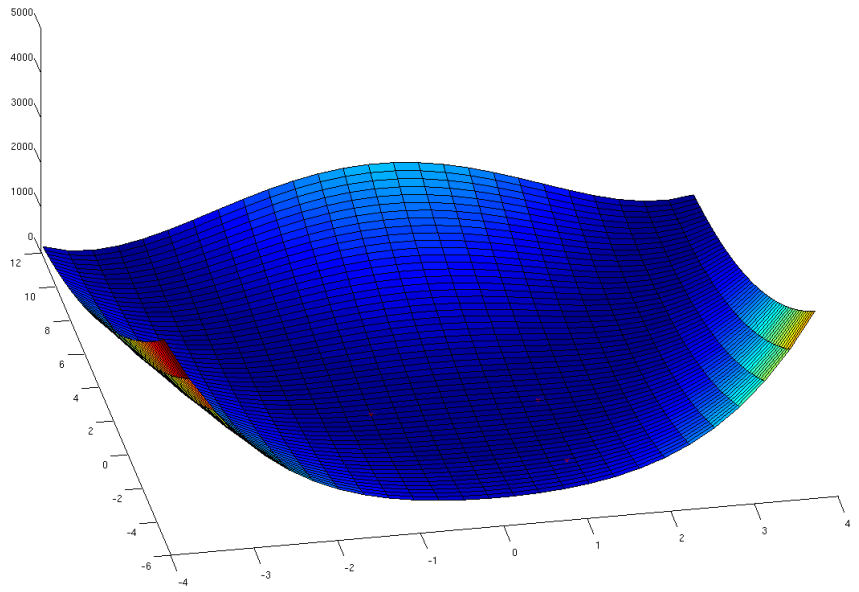


FIGURE 12 – Itérations de la méthode de Newton.



On remarque que la méthode de Newton converge en peu d'itération vers la solution. Cependant elle n'est pas rapide en pratique car elle demande d'inverser une matrice. De plus, elle demande à ce que la fonctionnelle soit deux fois dérivable, ce qui peut ne pas être le cas. De plus pour traiter de grandes données comme des images, la matrice hessienne sera de dimension $\sharp pixel \times \sharp pixel$ ce qui est trop gros pour être inversé. L'initialisation

de l'algorithme de newton est aussi un problème conséquent : on trouvera dans [3], des conditions initiales draconiennes.

1.6 Méthode de quasi-Newton.

La méthode de quasi-newton consiste à conserver un certain nombre de fois la même matrice hessienne afin d'éviter de recalculer l'inverse, puis on choisit à intervalle régulier de remettre à jour la matrice hessienne. On pourrait aussi prendre toujours la même matrice, voire même la matrice λId et on se retrouve alors avec la méthode du gradient à pas fixe.

On obtient les résultats suivants :

FIGURE 13 – Itérations de la méthode de quasi-Newton en 3-d.

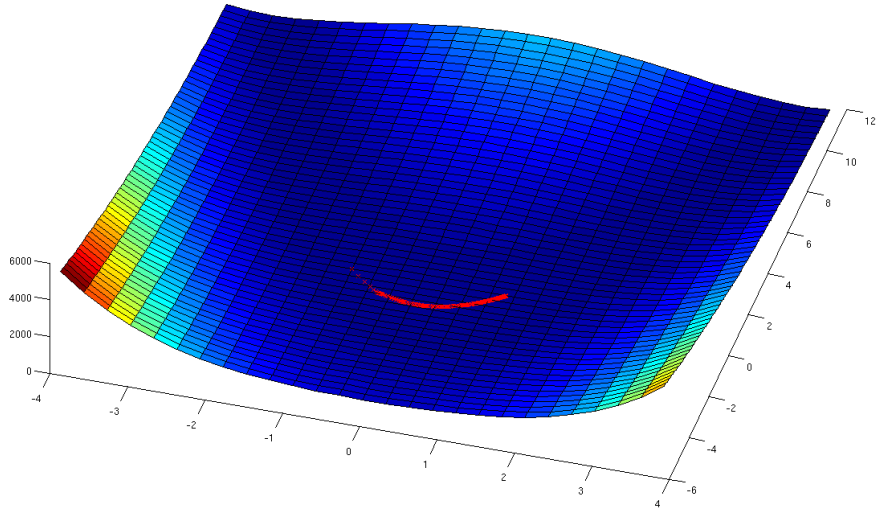
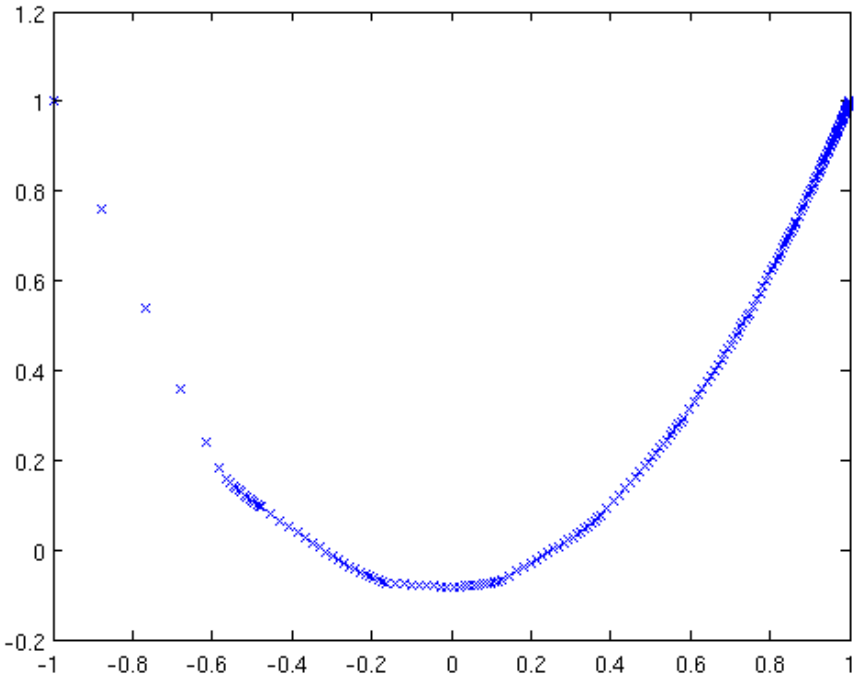


FIGURE 14 – Itérations de la méthode de quasi-Newton.



On observe que la direction privilégiée est modifiée lorsque la hessienne est mise à jour. La convergence est nettement plus lente que la vrai méthode de Newton.

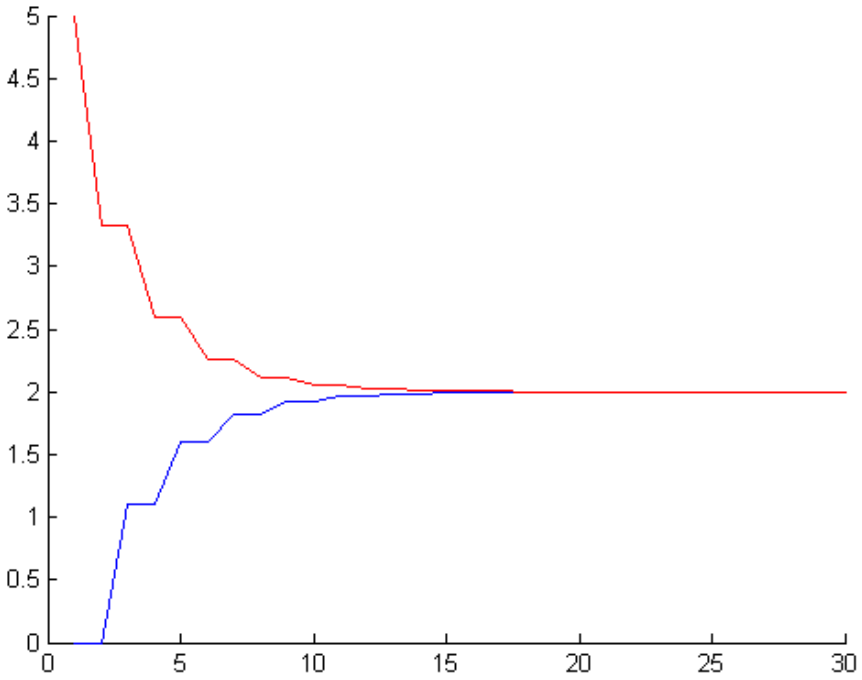
2 Algorithmes d'optimisation lisse avec contraintes.

2.1 Méthode de trichotomie (ou minimisation par dichotomie.)

On cherche le minimum d'une fonctionnelle $f : [a, b] \longmapsto \mathbb{R}$, convexe, ou bien le minimum sur un intervalle $[a, b]$ d'une fonctionnelle convexe et coercive. On applique pour cela un algorithme itératif de minimisation. On choisit deux points c et d de $[a, b]$ avec $a < c < d < b$, par choix. Si $f(c) < f(d)$, alors, pour des raisons évidentes de convexité de l'épigraphe de f, le minimum de f sur $[a, b]$ est contenu dans $[a, d]$ ce qui nous donne un nouvel intervalle. Sinon pour des raisons similaires, le nouvel intervalle est $[c, b]$. En itérant le procédé on réduit l'intervalle à volonté ce qui nous donne deux suites (les bornes de l'intervalle) convergeant monotonement vers le minimum de f sur $[a, b]$.

Si l'on applique cet algorithme à la recherche du minimum de $(x - 2)^2$ avec a_0 et $b_0 = 5$ on obtient les résultats suivants :

FIGURE 15 – Itérations de l'algorithme de trichotomie.



Ici on a utilisé une division de l'intervalle en 3. Ce n'est pas la meilleure solution. On peut choisir une division de l'intervalle de manière à ne pas recalculer les deux points intermédiaires mais seulement un des deux. Si on note L_k la longueur du $k-$ième intervalle, le fait de ne pas recalculer $f(c)$ et $f(d)$ implique nécessairement de supposer que

$$L_{k+2} = L_{k+1} + L_k. \tag{1}$$

Puisque l'on veut que la division de l'intervalle soit la même à chaque itérations on a de fait que $\frac{L_{k+1}}{L_k} = \gamma$ et alors, en divisant par L_{k+1} dans 1 , il vient que $\gamma = 1 + \frac{1}{\gamma}$ et donc $\gamma^2 - \gamma - 1 = 0$: donc γ vaut exactement le nombre d'or. Ainsi, il faut choisir $c = a + \left(1 - \frac{1}{\gamma}\right)|b-a|$ et $d = a + \frac{1}{\gamma}|b-a|$.

2.2 Relaxation contrainte sur un rectangle.

On résout ici un problème de minimisation sous contrainte de la forme :

$$J(u_0) = \inf_{u \in V} J(u)$$

avec V un ensemble convexe, fermé, borné, non vide, et J une fonctionnelle convexe sur V. Les théorèmes (voir [4] par exemple) nous assurent l'existence d'une unique solution.

On applique la méthode de relaxation à la fonctionnelle J considérée dans la section précédente sous la contrainte d'être sur un rectangle donné. Pour cela, on minimise coordonnée par coordonnée, comme pour la méthode de relaxation normale sauf que l'on minimise en se restreignant à être entre les bornes définies par le carré. On effectue la minimisation de chaque coordonnée par une méthode de minimisation en une dimension (descente de gradient par exemple ou encore trichotomie.)

2.3 Algorithmes de gradient projeté.

On considère toujours le même problème de minimisation sous contrainte. L'algorithme de gradient projeté nous donne une solution :

$$u_{k+1} \leftarrow P_V(u_k - \mu \nabla J(u_k))$$

où P_V est la projection sur l'ensemble V, et $\mu > 0$ un paramètre à régler. Cette méthode est assez artificielle car on ne peut pas toujours calculer la projection d'un point sur l'ensemble V. On donne ici des exemples où la méthode est applicable.

2.3.1 Minimisation L-2.

On cherche à minimiser la fonctionnelle $J(x) = ||b - Ax||^2$ sous contrainte que $x \geqslant 0$ [3] pour b et A un vecteur et un matrice donnés. On peut calculer explicitement le gradient de cette fonctionnelle :

$$\begin{aligned} \nabla J(x) &= \nabla \left[(b - Ax)^t (b - Ax) \right] \\ &= 2A^t Ax - 2A^t b \end{aligned}$$

ainsi que la projection sur l'ensemble $x \geqslant 0$: $(P(x_1, \cdots, x_n))_i = \max(0, x_i)$. L'algorithme devient :

$$u_{k+1} \leftarrow \max(0, u_k - \mu 2A^t A u_k - 2A^t b).$$

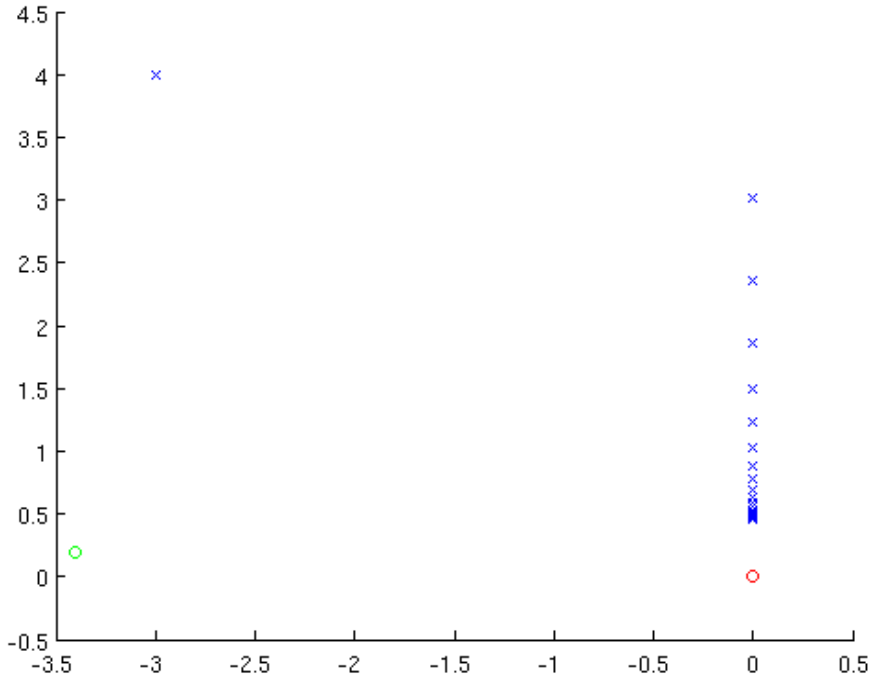
Application : On fait tourner l'algorithme avec la matrice

$$A = \begin{pmatrix} 1 & 2 \\ -1 & 3 \end{pmatrix}$$

et

$$\begin{pmatrix} -3 \\ 4 \end{pmatrix}$$

FIGURE 16 – Itérations de la méthode du gradient projeté pour la fonctionnelle $J(x) = ||b - Ax||^2$ sous contrainte que $x \geqslant 0$.



3. Cela signifie que si $x = (x_1, \cdots, x_n)$, alors $\forall i$ tel que $1 \leqslant i \leqslant n$, $x_i \geqslant 0$.

2.3.2 Projection sur un cylindre.

On reprend le problème de la minimisation L-2, sous contrainte de rester sur le cylindre (clairement convexe) $C := \{(x, y, z) \in \mathbb{R}^3 \text{ tels que } x^2 + y^2 = 1\}$, pour lequel on a une projection explicite :

$$P_C = \min\left(1, \left(\frac{(x, y)}{||(x, y, 0)||_2}, z\right)\right).$$

L'algorithme s'explicite alors :

$$(x, y, z) \leftarrow u_k - \mu 2A^t A u_k - 2A^t b$$

$$u_{k+1} \leftarrow \min\left(1, \left(\frac{(x, y)}{||(x, y, 0)||_2}, z\right)\right).$$

On teste avec la matrice

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 3 & 1 \\ 1 & 0 & 2 \end{pmatrix}$$

et

$$b = \begin{pmatrix} 3 \\ 1 \\ 3 \end{pmatrix}.$$

FIGURE 17 – Itérations de la méthode du gradient projeté sur le cylindre.

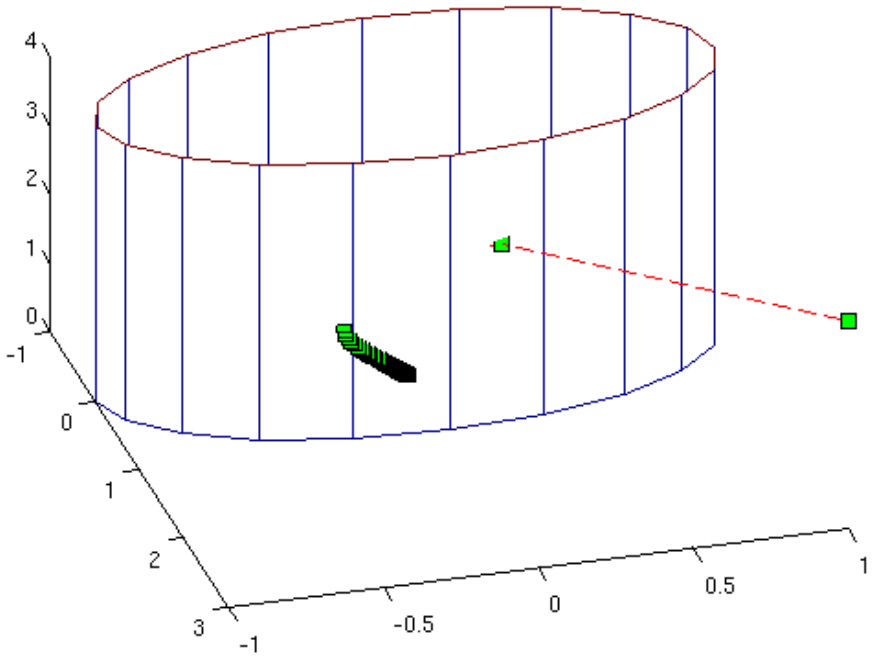
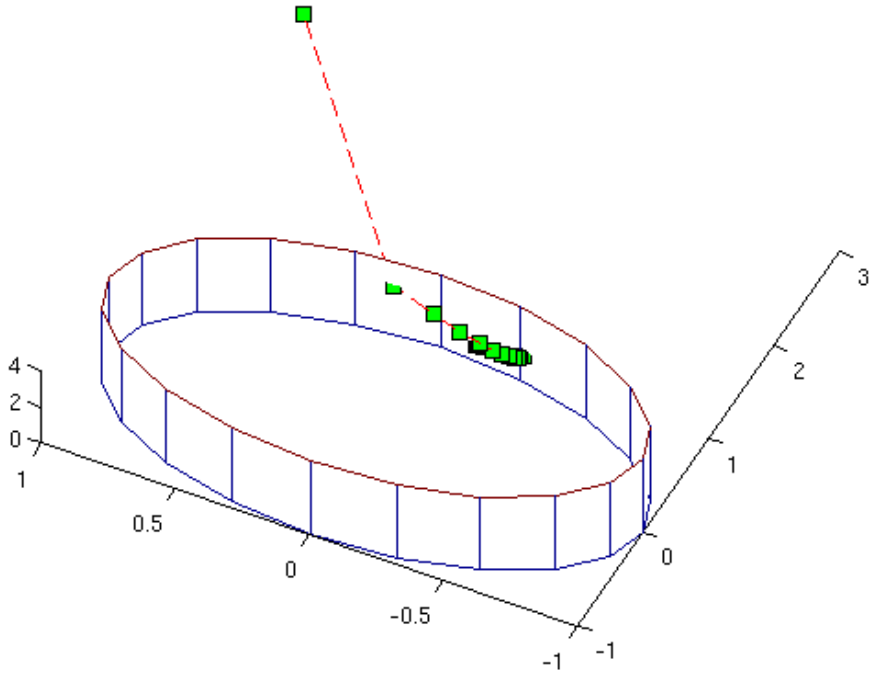


FIGURE 18 – Itérations de la méthode du gradient projeté sur le cylindre (vu du dessus.)



2.4 Méthode de gradient avec pénalisation.

On cherche toujours à minimiser sous contrainte :

$$J(u_0) = \inf_{u \in V} J(u).$$

On introduit le problème suivant :

$$J_\epsilon(u_0) = \inf_{u \in \mathbb{R}^n} J(u) + \frac{1}{\epsilon}\psi(u),$$

avec $\psi(v) > 0$, $\forall v \in \mathbb{R}^n \setminus V$ et $\psi(u) = 0$, $\forall u \in V$. On montre (dans [4]) que si $\epsilon \to 0$ alors, la solution de J_ϵ tend vers celle du premier problème.

Application.
On veut minimiser :

$$f(x, y) = x^2 + y^2 - 14x - 6y - 7$$

sous contraintes :

$$x + y \leqslant 2 \text{ et } x + 2y \leqslant 3.$$

On introduit la fonction de coût [4] :

$$\psi(x, y) = \max(x + y - 2, 0) + \max(x + 2y - 3, 0).$$

On cherche alors à minimiser sans contraintes :

$$J_\epsilon(x, y) = x^2 + y^2 - 14x - 6y - 7 + \frac{1}{\epsilon}\left[\max(x + y - 2, 0) + \max(x + 2y - 3, 0)\right],$$

4. Notons que l'on pourrait introduire la fonction différentiable (ψ ne l'est pas) :

$$\tilde{\psi}(x, y) = (\max(x + y - 2, 0))^2 + (\max(x + 2y - 3, 0))^2.$$

en faisant tendre ϵ vers 0. On va utiliser pour cela la méthode du gradient à pas fixe. Calculons pour mettre en œuvre cela, le gradient de J_ϵ :

$$\nabla J_\epsilon(x,y) = \begin{pmatrix} 2x - 14 + \frac{1}{\epsilon}\left\{ \begin{array}{c} 1 \text{ si } x+y-2 \geqslant 0 \\ 0 \text{ sinon} \end{array} \right\} + \frac{1}{\epsilon}\left\{ \begin{array}{c} 1 \text{ si } x+2y-3 \geqslant 0 \\ 0 \text{ sinon} \end{array} \right\} \\ 2y - 6 + \frac{1}{\epsilon}\left\{ \begin{array}{c} 1 \text{ si } x+y-2 \geqslant 0 \\ 0 \text{ sinon} \end{array} \right\} + \frac{1}{\epsilon}\left\{ \begin{array}{c} 1 \text{ si } x+2y-3 \geqslant 0 \\ 0 \text{ sinon} \end{array} \right\} \end{pmatrix}.$$

Ce qui s'implémente facilement. L'algorithme est un algorithme de gradient à pas fixe :

$$x_k \leftarrow x_{k-1} - \nu \nabla J_\epsilon(x_{k-1}).$$

Cela donne, pour deux initialisations différentes, les résultats suivants :

FIGURE 19 – Itérations de la méthode du gradient pénalisé initialisé en $(-1, 1)$.

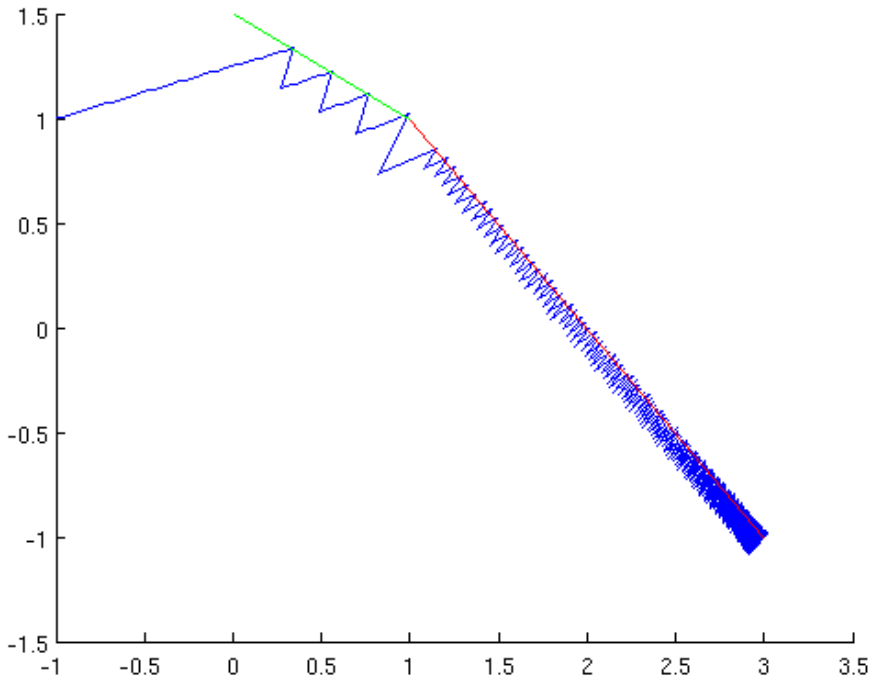
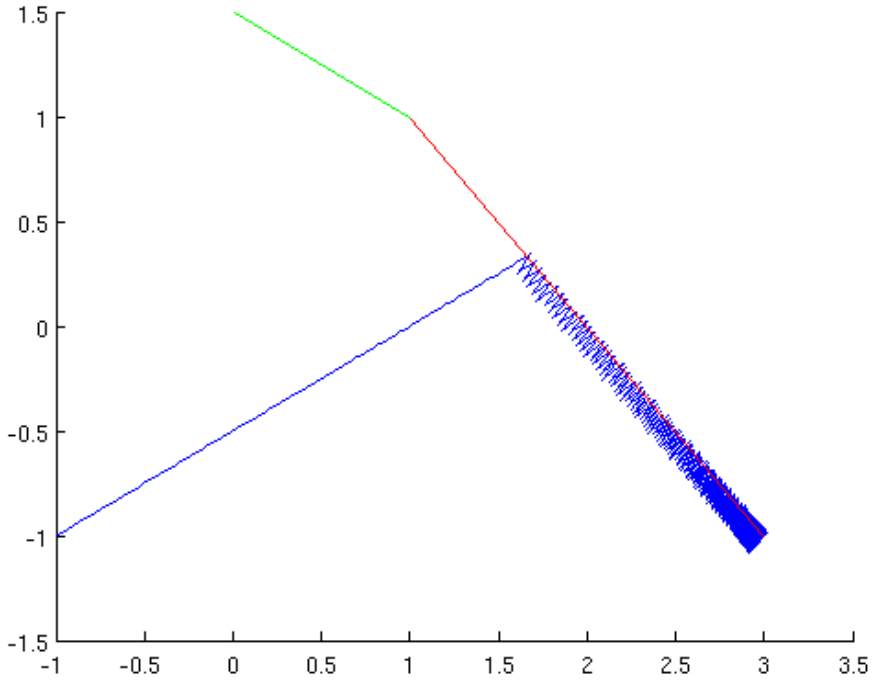


FIGURE 20 – Itérations de la méthode du gradient pénalisé initialisé en $(-1, -1)$.



On remarque que la contrainte rouge est la contrainte active.

3 Application de l'optimisation lisse au traitement d'image.

3.1 Norme L-2.

Soient T un opérateur linéaire de $\mathbb{R}^n$ dans $\mathbb{R}^m$, b un vecteur de $\mathbb{R}^m$, D un opérateur linéaire de $\mathbb{R}^n$ dans $\mathbb{R}^m$ correspondant à un filtre passe-haut, (on prendra un opérateur de dérivation discrète en pratique), λ un réel positif, on considère le problème suivant :

$$J(u_0) = \inf_{u \in \mathbb{R}^n} \lambda ||Du||_2^2 + ||Tu - b||_2^2.$$

On cherche à mettre en place un des algorithmes vus en section 1 pour résoudre ce problème. On va se consacrer au gradient à pas fixe.

Il faut donc calculer le gradient de J :

$$\begin{aligned} \nabla J(u) &= \nabla \left[\lambda ||Du||_2^2 + ||Tu - b||_2^2\right] \\ &= \left[\lambda u^t D^t Du + u^t T^t Tu - 2u^t T^t b + ||b||_2^2\right]. \\ &= \lambda 2 D^t Du + 2T^t Tu - 2T^t b \end{aligned}$$

Ce calcul nous permet d'avoir, si $\lambda D^t D + T^t T$ est définie positive (ce que l'on suppose facilement, dans le but d'imposer un minimum à notre fonctionnelle,) les *équations normales* :

$$u = (\lambda D^t D + T^t T)^{-1} T^t b.$$

3.1.1 Application en restauration.

Dans le cas de la restauration la donnée b est une image bruitée, et il s'agit de résoudre le problème de minimisation avec l'opérateur $T = Id$. $T^t = T = Id$ et alors Tu s'obtient par $Tu = T^t u = \delta_0 * u$. L'opérateur D^t s'obtient à partir de D en faisant une symétrie verticale et horizontale des colonnes. En effet :

Posons, si $x = (x_1, \cdots, x_n)$, $Dx = (x_1 - x_2, x_2 - x_3, \cdots, x_{n-1} - x_n, x_n)$.
Alors si $u = (u_1, \cdots, u_n)$ et $v = (v_1, \cdots, v_n)$:

$$\begin{aligned}\langle Du|v\rangle &= \langle (u_1 - u_2, u_2 - u_3, \cdots, u_{n-1} - u_n, u_n)|(v_1, \cdots, v_n)\rangle \\ &= (u_1 - u_2)v_1 + (u_2 - u_3)v2 + \cdots + u_n v_n \\ &= \langle (u_1, \cdots, u_n)|(v_1, v_2 - v_1, \cdots, v_n - v_{n-1})\rangle \\ &= \left\langle u, D^t v\right\rangle\end{aligned}$$

Donc si $D = \delta_0 - \delta_1$, $D^t = \delta_1 - \delta_0$. Ce que l'on peut généraliser à deux dimensions. En Matlab, cela donnera [5] :

```
Dt=rot90(D,2);
```

Notons au passage, que cette commande est parfaitement équivalente, puisque D est une matrice réelle, dont la diagonale, les sous-diagonales et les sur-diagonales sont homogènes, à :

```
Dt=D';
```

FIGURE 21 – Image bruitée par un bruit additif gaussien d'écart-type 30.

5. Car la composée de deux symétrie d'axes Δ_1 et Δ_2 avec $\widehat{\Delta_1\Delta_2} = \alpha \neq 0$ est une rotation de centre $\Delta_1 \cup \Delta_2$ et d'angle 2α.

FIGURE 22 – Image débruitée avec $\lambda = 1.2$.

On donne le SNR [6] :

avant débruitage	6.4715
après débruitage	9.5465

.

FIGURE 23 – Image débruitée avec $\lambda = 0.1$.

3.1.2 Application en déconvolution.

En déconvolution l'image considérée est convoluée et bruitée. L'opérateur T est donc un opérateur de convolution. Sa transposée se calcule comme pour l'opérateur D. Ainsi $Tu = T * u$ et $T^t u = T^t * u$ avec T^t obtenu par :

6. Donné par la formule $SNR = 10 \log_{10} \left(\frac{||s||_2}{||u - s||_2} \right)$, ou u est l'image débruitée, et s l'image non bruitée originale.

```
Tt=T';
```

FIGURE 24 – Noyau de convolution.

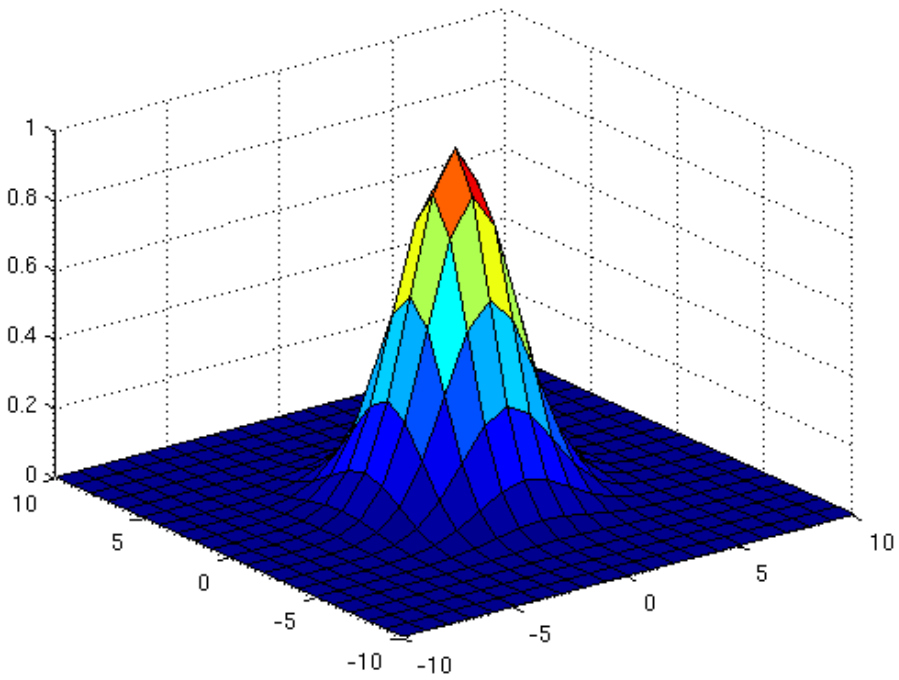


FIGURE 25 – Image convoluée et bruitée.

FIGURE 26 – Résultat de l'inversion.

3.1.3 Application en Inpainting.

En inpainting, l'opérateur devient Id sur les parties non manquantes et 0 sur les parties manquantes. Notons M cet opérateur [7].

Théorème 1
L'opérateur transposé de M est M lui même :

$$M^t = M.$$

Preuve En effet si $u = (u_1, \cdots, u_n)$ et $v = (v_1, \cdots, v_n)$:

$$\begin{aligned}
\langle Tu|v\rangle &= \left\langle \left(\left\{ \begin{array}{c} u_1 \text{ si valeur masquée} \\ 0 \text{ sinon} \end{array} \right\}, \cdots, \left\{ \begin{array}{c} u_n \text{ si valeur masquée} \\ 0 \text{ sinon} \end{array} \right\} \right) | (v_1, \cdots, v_n) \right\rangle \\
&= \left\{ \begin{array}{c} u_1 \text{ si valeur masquée} \\ 0 \text{ sinon} \end{array} \right\} v_1 + \cdots + \left\{ \begin{array}{c} u_n \text{ si valeur masquée} \\ 0 \text{ sinon} \end{array} \right\} v_n \\
&= \left\langle (u_1, \cdots, u_n) | \left(\left\{ \begin{array}{c} v_1 \text{ si valeur masquée} \\ 0 \text{ sinon} \end{array} \right\}, \cdots, \left\{ \begin{array}{c} v_n \text{ si valeur masquée} \\ 0 \text{ sinon} \end{array} \right\} \right) \right\rangle \\
&= \langle u, D^t v \rangle
\end{aligned}$$

Ce qui nous donne les moyens de calculer Tu et $T^t u$: il s'agit d'une multiplication, par le masque défini, coordonnée par coordonnée.

Cela donne les résultats suivants :

7. M comme *masque* ou *masquage*.

FIGURE 27 – Image masquée 75% de données manquantes.

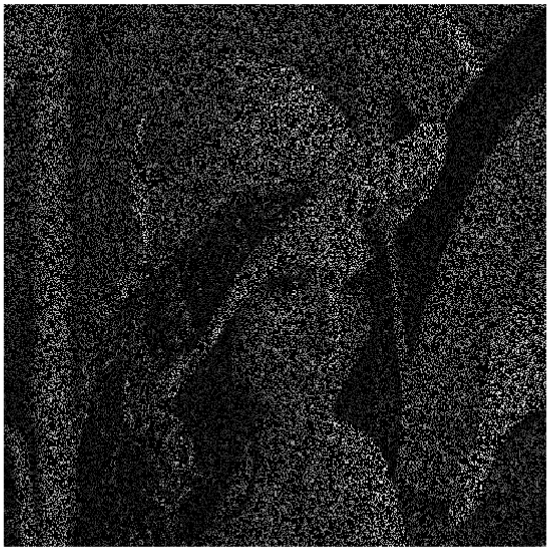


FIGURE 28 – Image inpaintée.

L'algorithme restitue bien les données perdues. Néanmoins, il floute un peu.

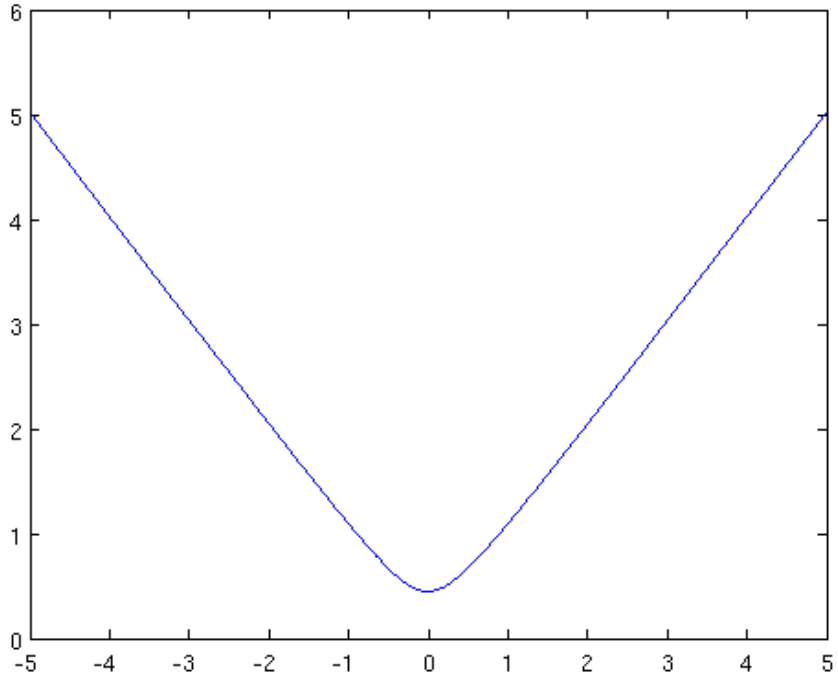
3.2 Norme L-1-L-2.

On choisit désormais le critère L-1-L-2 : Sous les même hypothèses que précédemment, on considère le problème suivant :

$$J(u_0) = \inf_{u \in \mathbb{R}^n} \lambda ||Du||_{\text{L-1-L-2},\epsilon} + ||Tu - b||_2^2.$$

Avec $||x||_{\text{L-1-L-2},\epsilon} = \sqrt{||x||_2^2 + \epsilon}$, dont on dessine une représentation en une dimension.

FIGURE 29 – Fonction L-1-L-2 en une dimension.



Déterminons, afin d'appliquer l'algorithme du gradient à pas fixe, le gradient de cette fonction :

$$\nabla||Du||_{\text{L-1-L-2},\epsilon} = \frac{1}{2}\frac{\nabla||Du||_2^2}{\sqrt{||Du||_2^2+\epsilon}}.$$

On dispose donc de tout les moyens nécessaires pour mettre en place l'algorithme du gradient pour la minimisation de ce critère.

3.2.1 Application en restauration.

Les conditions sont identiques à celle de la norme L-2. Les résultats sont les suivants :

FIGURE 30 – Image bruitée.

FIGURE 31 – Image restaurée.

On peut comparer la qualité de ces résultats avec ceux obtenus avec la norme−2 seule :

	Norme L-2	Norme L-1-L-2
SNR avant débruitage	6.4715	6.4663
SNR après débruitage	9.5465	10.4629

.

3.2.2 Application en déconvolution.

Avec le même opérateur de convolution et le même niveau de bruit.

FIGURE 32 – Image convoluée et bruitée.

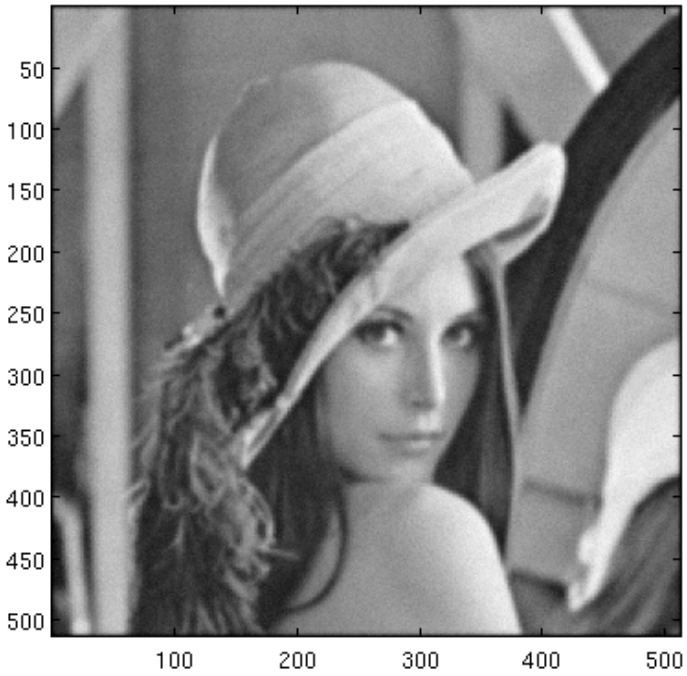
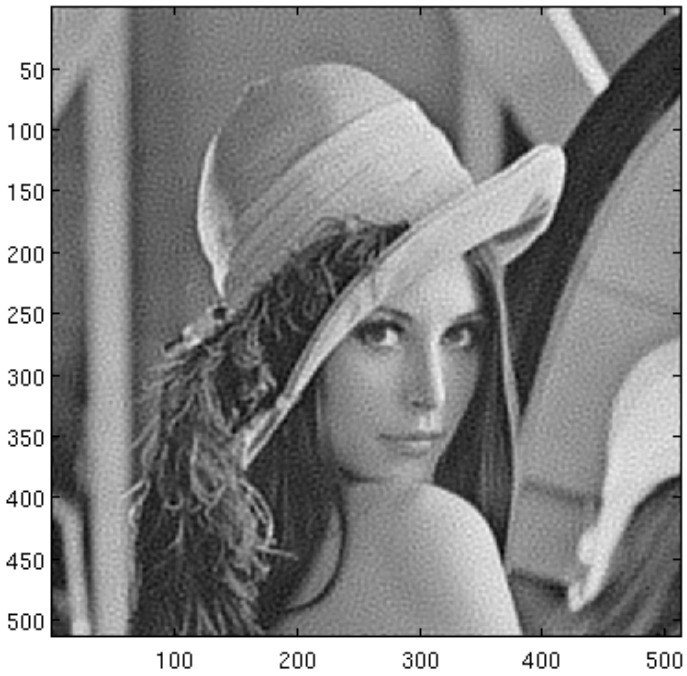


FIGURE 33 – Image restaurée.



3.2.3 Application en Inpainting.

L'opérateur est l'opérateur de masquage, mettant à zéro 75% des pixels à 0 de façon aléatoire.

FIGURE 34 – Image originale masquée.

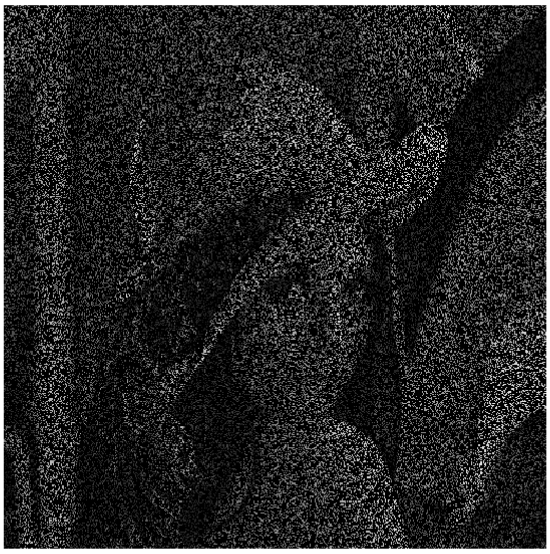
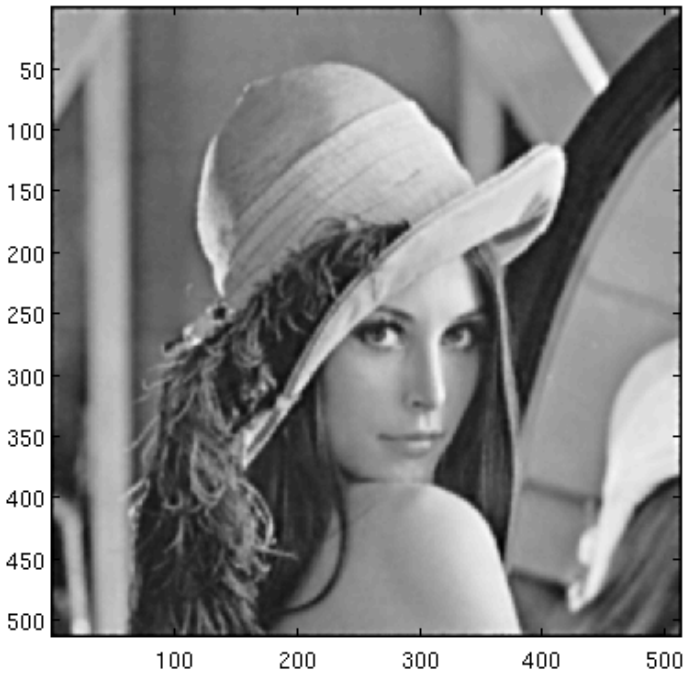


FIGURE 35 – Image impaintée en accord avec le critère L-1-L-2.



Pour cette dernière application, il est bon de noter que le paramètre $\lambda_{\text{L-1-L-2},\epsilon}$ est grand devant le paramètre λ que l'on utilisait pour la norme L-2 (notons le λ_2) car on vérifie en pratique que $||Du||_2^2 \sim 10^8$ donc le rapport entre $||Du||_2^2$ et $||Du||_{\text{L-1-L-2},\epsilon}$ est d'environ 10^4 ce qui nous donne donc l'ordre de grandeur de $\dfrac{\lambda_{\text{L-1-L-2},\epsilon}}{\lambda_2} \sim 10^4$. Cela nous indique qu'il faut "gonfler" le paramètre λ, ce qui se vérifie parfaitement en pratique. On a ici dû utiliser une multiplication du paramètre par 2000 pour obtenir un résultat correct.

4 La minimisation non lisse.

4.1 Notion de sous-différentielle.

Définition (La sous-différentielle.)
Soit Γ_0 l'ensemble des fonctions convexes de $\mathbb{R}^N \mapsto \mathbb{R} \cup \{+\infty\}$ semi-continue inférieurement, i.e.

$$\Gamma_0 := \{f \text{ tq } \forall x_0 \in \mathbb{R}^N \ , \ \forall \epsilon > 0 \ , \ \exists V \text{ voisinage de } x_0 \ , \ \forall x \in V : f(x) \leqslant f(x_0) - \epsilon\} \ .$$

On définit [8] la sous-différentielle au point x, l'ensemble

$$\partial f(x) := \{u \in \mathbb{R}^N \text{ tq } \forall y \in \mathbb{R}^N \ , \ \langle y - x | u \rangle + f(x) \leqslant f(y)\} \ .$$

Exemple :
L'application $f : x \mapsto |x|$ est convexe, continue donc semi-continue inférieurement. Et on a : $\partial f(0) = [-1, 1]$, $\partial f(x) = \{1\}$ si $x > 0$ et $\partial f(x) = \{-1\}$ si $x < 0$.

Théorème 2 (Caractérisation d'un minimum.)
Soit J une fonctionnelle convexe semi-continue inférieurement, J admet un minimum en u_0 si et seulement si $0 \in \partial J(u_0)$.

Preuve On trouvera la preuve dans [10].

8. [8]

4.2 Projection et opérateurs proximaux.

Soit C un ensemble connexe fermé non vide. On définit par

$$1_C(x) := \begin{cases} 0 \text{ si } x \in C \\ +\infty \text{ sinon.} \end{cases}$$

La projection $P_C(x)$ de l'élément x sur le convexe fermé C est la solution u_0 du problème

$$\min_{y \in \mathbb{R}^N} 1_C(y) + \frac{1}{2}||x-y||^2.$$

On a que 1_C est une fonction de Γ_0.

On étend donc cette notion en définissant l'opérateur proximal d'un fonction semi-continue inférieurement :

Définition (Opérateur proximal.)
Soient $f \in \Gamma_0$, $x \in \mathbb{R}^N$ le problème suivant a une unique solution :

$$\min_{y \in \mathbb{R}^N} f(y) + \frac{1}{2}||x-y||^2,$$

notée $\text{prox}_f(x)$ appelé l'*opérateur proximal* de f.

Proposition 1
Soit $f \in \Gamma_0$. L'opérateur proximal de f est caractérisé par :

$$\forall (x,y) \in \mathbb{R}^N \times \mathbb{R}^N \ , \ y = prox_f(x) \Leftrightarrow x - y \in \partial f(y).$$

Remarque : 1 Dans ce cas particulier, l'opérateur proximal peut prendre la notation suivante :

$$z := \text{prox}_f(x) = (I_d + \partial f)^{-1}(x).$$

Posons pour cela z tel que

$$f(z) + \frac{1}{2}||x-z||^2 = \min_{y \in \mathbb{R}^N} f(y) + \frac{1}{2}||x-y||^2.$$

Alors on a $z := \text{prox}_f(x)$ et donc $x - z \in \partial f(z)$. Ou encore $x \in z + \partial f(z)$. Ce que l'on note

$$x = (I_d + \partial f(z)).$$

On note cela formellement :

$$z = (I_d + \partial f)^{-1}(x).$$

Remarque : 2 Dans le cas où f est différentiable il vient :

$$\forall (x,y) \in \mathbb{R}^N \times \mathbb{R}^N \ , \ y = \text{prox}_f(x) \Leftrightarrow x - y = \nabla f(y).$$

4.3 À propos de la transformation de Legendre-Fenchel.

Définition
On définit (dans [8] par exemple) la transformée de Legendre-Fenchel par $J^*(v) = \sup_u \langle u|v \rangle - J(u)$.

Proposition 2
Soit J une fonctionnelle homogène de degré 1 (i.e. $\forall u \in \mathbb{R}^N$ et $\forall \lambda > 0$ on a $J(\lambda u) = \lambda J(u)$.) Alors l'analyse convexe nous dit que

$$J^*(v) = 1_K(v).$$

Corollaire 1
Soit J une fonctionnelle homogène de degré 1, alors $J^{**}(u) = J(u)$.

Corollaire 2
Soit J une fonctionnelle homogène de degré 1, alors $J(v) = \sup_{u \in K} \langle u|v \rangle$.

Preuve En effet, par le corollaire 1 alors :

$$J(u) = J^{**}(u) = \sup_{v \in \mathbb{R}^N} \{\langle u|v \rangle - J^*(u)\} ,$$

mais d'après la proposition 2 alors :

$$J(u) = J^{**}(u) = \sup_{v \in \mathbb{R}^N} \{\langle u|v \rangle - 1_K(v)\} ,$$

et donc le sup se restreint à $v \in K$. Donc

$$J(v) = \sup_{u \in K} \langle u|v \rangle .$$

4.4 Utilisation de la transformée de Legendre-Fenchel pour la sous-différentielle.

Théorème 3
Soit $f \in \Gamma_0$ alors :

$$u \in \partial f(v) \Longleftrightarrow v \in \partial f^*(u).$$

Preuve On la trouvera dans [10], page 48.

Soit $g \in \mathbb{R}^{N \times N}$ une image, $\lambda > 0$. On pose $||u||^2 = \langle u|u \rangle$. On veut minimiser

$$\min_{u \in X} \frac{||u-g||_2^2}{2\lambda} + J(u).$$

L'équation de Legendre s'écrit

$$0 \in u - g + \lambda \partial J(u),$$

où ∂J est la sous-différentielle définie plus haut.

Ce qui est équivalent, par le théorème 3 , à :

$$u \in \partial J^* \left(\frac{g-u}{\lambda} \right),$$

ou

$$\frac{g}{\lambda} \in \frac{g-u}{\lambda} + \frac{1}{\lambda} \partial J^* \left(\frac{g-u}{\lambda} \right). \tag{2}$$

Théorème 4
Si $J \in \Gamma_0$ alors :

$$w = (I + s\partial J)^{-1}(v) \Leftrightarrow w \text{ minimise } \frac{||w-v||^2}{2} + sJ(w).$$

Théorème 5
Si $J^* = 1_K$, alors l'opérateur $(I + s\partial J^*)^{-1}$ est la projection sur l'ensemble convexe fermé K, notée π_K.

Preuve (de 4 et 5.)

$$\begin{aligned} w = (I + s\partial J^*)^{-1}(v) &\Leftrightarrow v = w + s\partial J^*(w) \\ &\Leftrightarrow 0 \in w - v + s\partial J^*(w) \\ &\Leftrightarrow w \text{ minimise } \frac{||w-v||^2}{2} + sJ^*(w) \end{aligned}$$

avec $J^* = 1_K$.

Avec de telle notations, 2 devient

$$u = g - \lambda \left(I + \frac{1}{\lambda} \partial J^* \right)^{-1} \left(\frac{g}{\lambda} \right).$$

Et la solution de notre problème initial devient $u = g - \pi_K(g)$.

4.5 Application de la transformation de Legendre-Fenchel à la minimisation de la variation totale.

Définition
Posons

$$\nabla u_{i,j} = \begin{pmatrix} \begin{cases} u_{i+1,j} - u_{i,j} \text{ si } i < N, \\ 0 \text{ si } i = N, \end{cases} \\ \begin{cases} u_{i,j+1} - u_{i,j} \text{ si } j < N, \\ 0 \text{ si } j = N. \end{cases} \end{pmatrix}.$$

On pose également :

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

Lemne 1 (Dualité gradient-divergence.)

$$\langle -divp | u \rangle = \langle p | \nabla u \rangle .$$

La démonstration est uniquement calculatoire et tombe juste car la divergence a été définie pour cela.

Cette formule est l'équivalent discret d'une formule découlant de la formule de Stokes appelée parfois formule du gradient. Soit $f \in C^1$, $\overrightarrow{A}$ un champ de vecteur :

$$\int_{\mathbb{R}^n} f \mathrm{div}\, \overrightarrow{A} \mathrm{d}\lambda_n(x) = -\int_{\mathbb{R}^n} \nabla f \cdot \overrightarrow{A} \mathrm{d}\lambda_n(x)$$

On veut minimiser

$$J(u) = \sum_{1 \leqslant i,j \leqslant N} \sqrt{\left(\nabla u^{(1)}_{i,j}\right)^2 + \left(\nabla u^{(2)}_{i,j}\right)^2}. \tag{3}$$

Pour minimiser $J(u)$ on utilise le résultat suivant :

Proposition 3

$$\begin{aligned} J(u) &:= \sum_{1 \leqslant i,j \leqslant N} \sqrt{\left(\nabla u^{(1)}_{i,j}\right)^2 + \left(\nabla u^{(2)}_{i,j}\right)^2} \\ &= \sup_{\{p \ tq \ |p_{i,j}| \leqslant 1\}} \langle p | \nabla u \rangle . \end{aligned}$$

Preuve En effet, on a par Cauchy-Schwarz :

$$p^{(1)}_{i,j} \nabla u^{(1)}_{i,j} + p^{(2)}_{i,j} \nabla u^{(2)}_{i,j} \leqslant |\nabla u_{i,j}||p_{i,j}|,$$

et si $|p_{i,j}| = 1$, on a

$$p^{(1)}_{i,j} \nabla u^{(1)}_{i,j} + p^{(2)}_{i,j} \nabla u^{(2)}_{i,j} \leqslant |\nabla u_{i,j}|,$$

d'où

$$\sum_{i,j} p^{(1)}_{i,j} \nabla u^{(1)}_{i,j} + p^{(2)}_{i,j} \nabla u^{(2)}_{i,j} \leqslant \sum_{i,j} |\nabla u_{i,j}|$$

et

$$\sup_{\{p \text{ tq } |p_{i,j}| \leqslant 1\}} \sum_{i,j} p^{(1)}_{i,j} \nabla u^{(1)}_{i,j} + p^{(2)}_{i,j} \nabla u^{(2)}_{i,j} \leqslant \sum_{i,j} |\nabla u_{i,j}|.$$

Et on obtient le cas d'égalité, si $|\nabla u_{i,j}| \neq 0$ grâce à $p_{i,j} = \dfrac{\nabla u_{i,j}}{|\nabla u_{i,j}|}$. Si $|\nabla u_{i,j}| = 0$, l'égalité est immédiatement vérifiée.

Corollaire 3
On déduit du lemne 1 que

$$J(u) = \sup_{\{divp \ tq \ |p_{i,j}| \leqslant 1 \ , \ \forall i,j=1,\cdots,N\}} \langle p | u \rangle .$$

4.6 Application au débruitage d'image.

4.6.1 Considérations théoriques en vue de la construction de l'algorithme.

Le modèle ROF tient son nom de celui de ses inventeurs : Rudin, Osher et Fatemi : Soit $g \in \mathbb{R}^{N\times N}$ une image, $\lambda > 0$. On pose $||u||^2 = \langle u|u\rangle$. On veut minimiser

$$\min_{u\in X} \frac{||u-g||_2^2}{2\lambda} + J(u) \ ,$$

avec J la variation totale dont on rappelle l'expression :

$$J(u) = \sum_{1\leqslant i,j\leqslant N} \sqrt{\left(\nabla u_{i,j}^{(1)}\right)^2 + \left(\nabla u_{i,j}^{(2)}\right)^2}.$$

D'après le travail fait dans 4.4 , la solution de notre problème est sous la forme d'une projection $u = g - \pi_K(g)$.

Chambolle, dans son article [1], explique qu'il est facile de programmer cette projection en dimension 1. Et il propose un algorithme pour calculer la projection en dimension 2. Il s'agit de résoudre le problème suivant :

$$\min\left\{||\lambda \text{div} p - g||^2 \text{ sous contrainte } ||p_{i,j}||_2^2 \leqslant 1\right\}.$$

Les relations de Karush-Kuhn-Tucker s'écrivent ainsi : Il existe $\alpha_{i,j}$ tels que :

$$\begin{cases} \nabla(\lambda \text{div} p - g)_{i,j} + \alpha_{i,j} p_{i,j} = 0 \\ \alpha_{i,j} \geqslant 0 \\ \alpha_{i,j}(||p_{i,j}||_2^2 - 1) = 0 \end{cases} .$$

On a deux éventualités : $\alpha_{i,j} > 0$ et $||p_{i,j}||_2 = 1$; ou bien $\alpha_{i,j} = 0$ et $||p_{i,j}||_2 < 1$. Dans les deux cas $\alpha_{i,j} = |\nabla(\lambda \text{div} p - g)_{i,j}|$. On obtient ainsi l'équation à point fixe :

$$p_{i,j}^{(n+1)} = p_{i,j}^{(n)} + \tau\left((\nabla(\lambda \text{div} p - g))_{i,j} - |(\nabla(\lambda \text{div} p - g))_{i,j}| p_{i,j}^{(n+1)}\right)$$

avec $\tau > 0$. Et de façon explicite :

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

La convergence de cet algorithme est démontré dans [1].

4.6.2 Mise en œuvre.

Si l'on considère l'algorithme tel quel, on peut faire du débruitage en projetant l'image bruitée sur K.

Choisissons une image bruitée et appliquons l'algorithme.

FIGURE 36 – Image bruitée par un bruit additif gaussien d'écart-type 10.

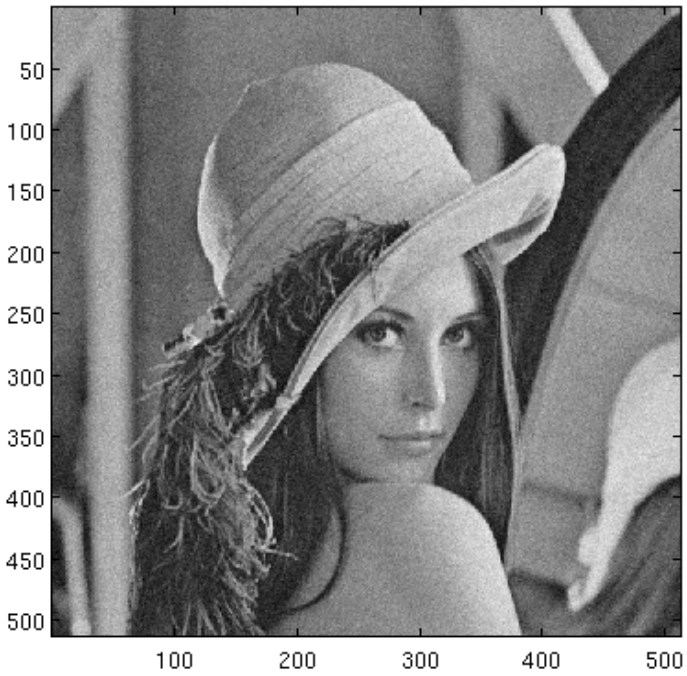


FIGURE 37 – Image débruitée par une simple projection.

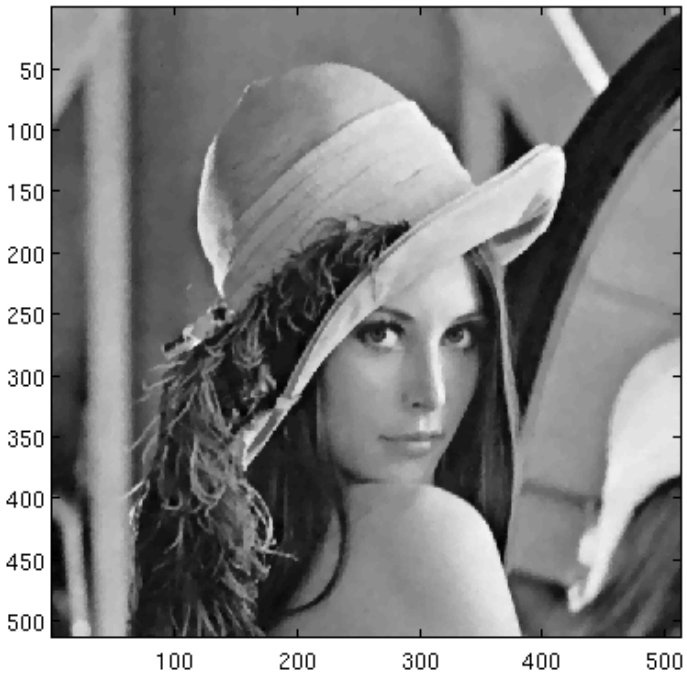


FIGURE 38 – Image bruitée par un bruit additif gaussien d'écart-type 30.

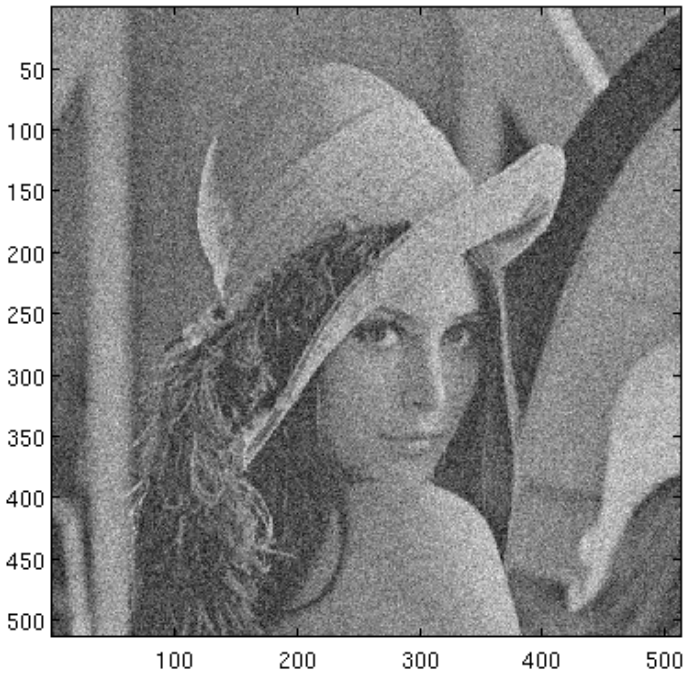
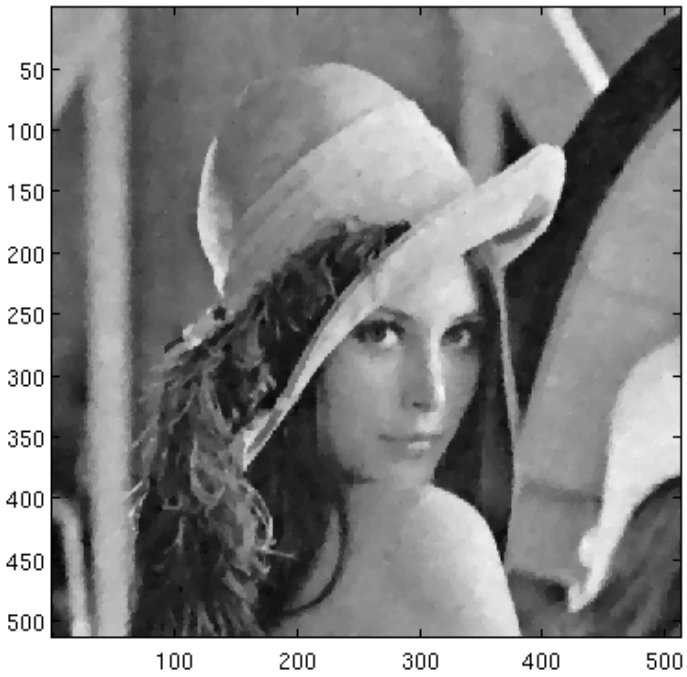


FIGURE 39 – Image débruitée par une simple projection.



4.6.3 L'algorithme non supervisé de Chambolle.

Néanmoins, l'algorithme ne retourne pas une image très bien débruitée car le paramètre λ n'est pas forcément bien choisit. Chambolle propose donc un algorithme spécialisé dans le débruitage d'image bruitée par un bruit additif gaussien dont on est censé connaitre l'écart-type σ. On pose le problème suivant :

$$\min \left\{ J(u) \text{ sous contrainte } ||u-g||^2 = N^2\sigma^2 \right\}. \tag{4}$$

Posons d'abord $\overline{g}$ la valeur moyenne des pixels de l'image $g_{i,j}$, $1 \leqslant i,j \leqslant N$. On suppose que $0 \leqslant N\sigma \leqslant ||g-\overline{g}||$.

Initialisation :

- $\lambda_0 > 0$
- $v_0 = \pi_{\lambda_0 K}(g)$
- $f_0 = ||v_0||$

Itération :

- $\lambda_{n+1} \leftarrow \dfrac{N\sigma}{f_n}\lambda_n$
- $v_{n+1} \leftarrow \pi_{\lambda_{n+1}K}(g)$
- $f_{n+1} \leftarrow ||v_{n+1}||$.

Théorème 6
Lorsque $n \to +\infty$ alors $f_n \to N\sigma$ tandis que $g - v_n$ converge vers la solution de 4.

La preuve de ce théorème est donnée dans [1]. Il n'est pas très rapide en l'état car il nécessite de calculer de nombreuses fois la projection $\pi_{\lambda_n K}(g)$ qui est assez lente. L'idée qui permet d'accélérer l'algorithme est de ne pas initialiser le calcul de la projection à zéro à chaque fois mais de reprendre les deux dernières valeurs sur lesquelles on était arrivé. Cela accélère nettement l'algorithme. On arrive ainsi aux résultats suivants en environ 12 secondes :

FIGURE 40 – Image bruitée par un bruit additif gaussien d'écart-type 30.

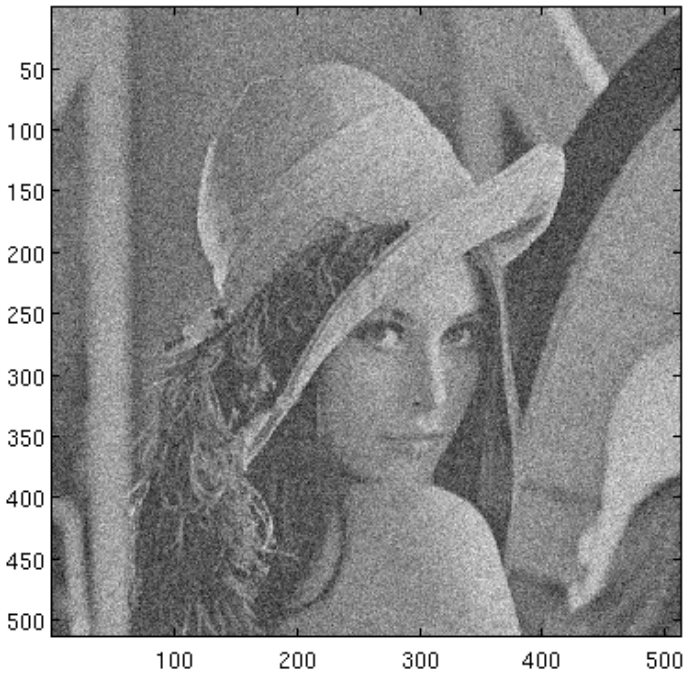


FIGURE 41 – Image débruitée par l'algorithme décrit ci-dessus.

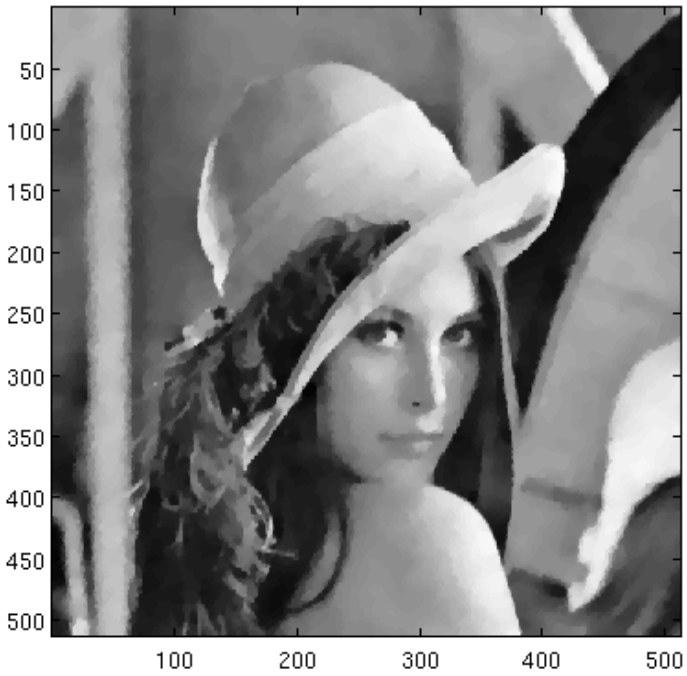
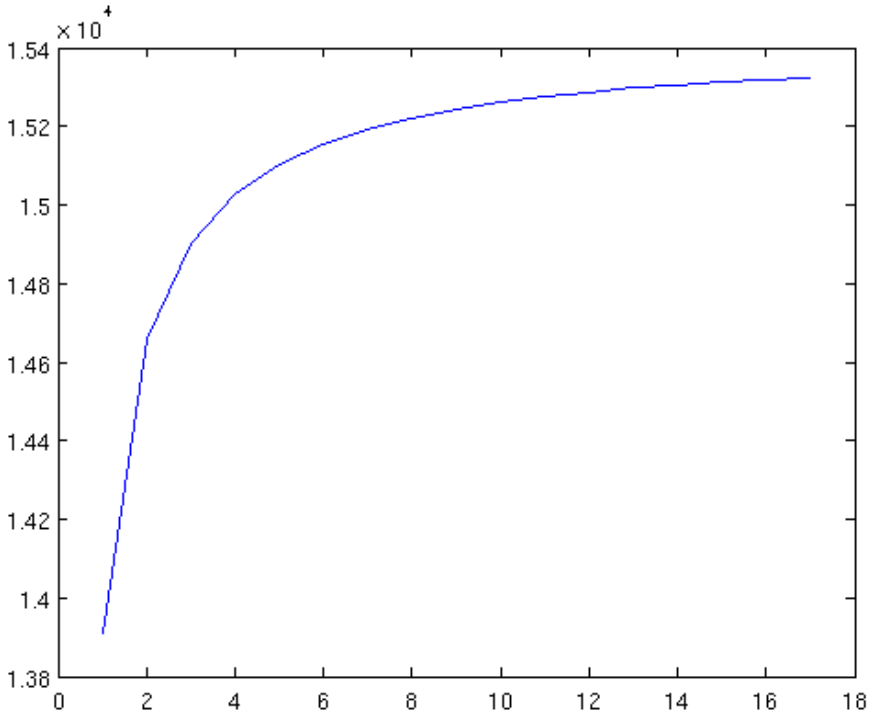


FIGURE 42 – Courbe de l'évolution de f_n en fonction du temps.



Pour cette dernière figure, remarquons que $N\sigma = 512 \times 30 = 15360$.
Avec une autre image, en environ 6 secondes :

FIGURE 43 – Image originale.

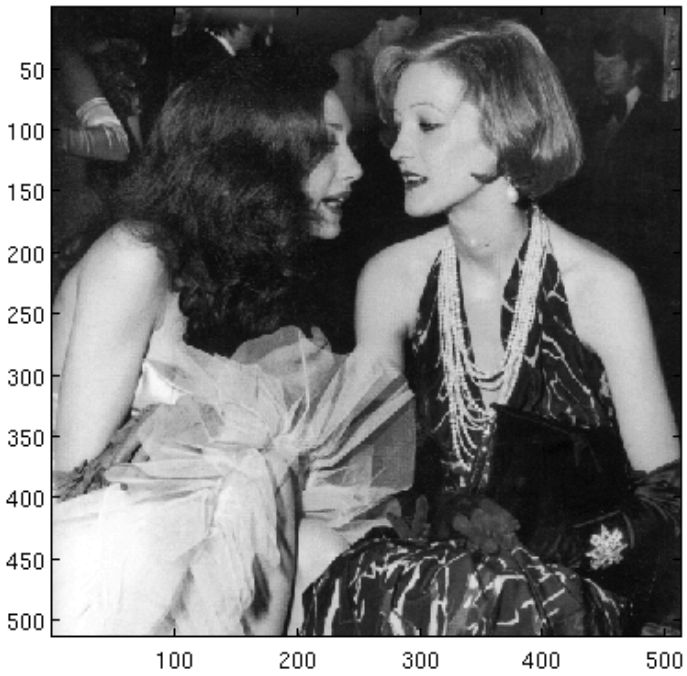


FIGURE 44 – Image bruitée par un bruit additif gaussien d'écart-type 30.

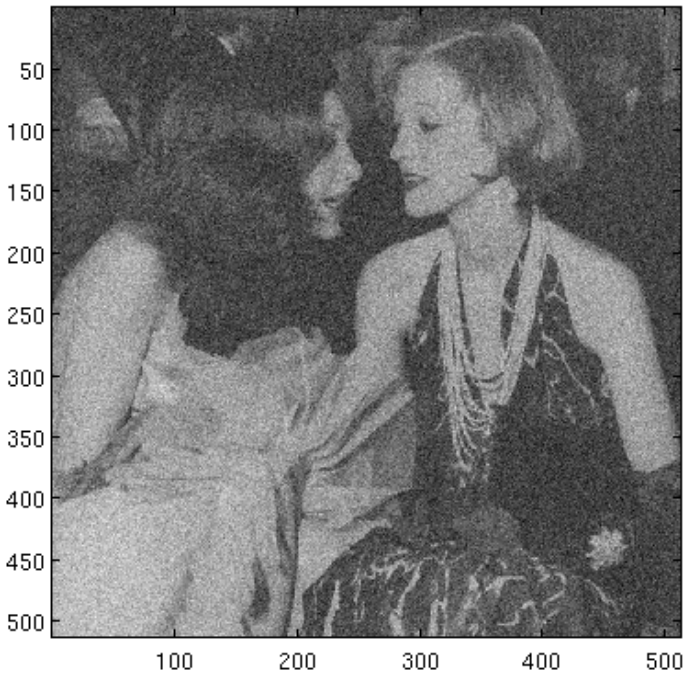


FIGURE 45 – Image débruitée par l'algorithme décrit ci-dessus.

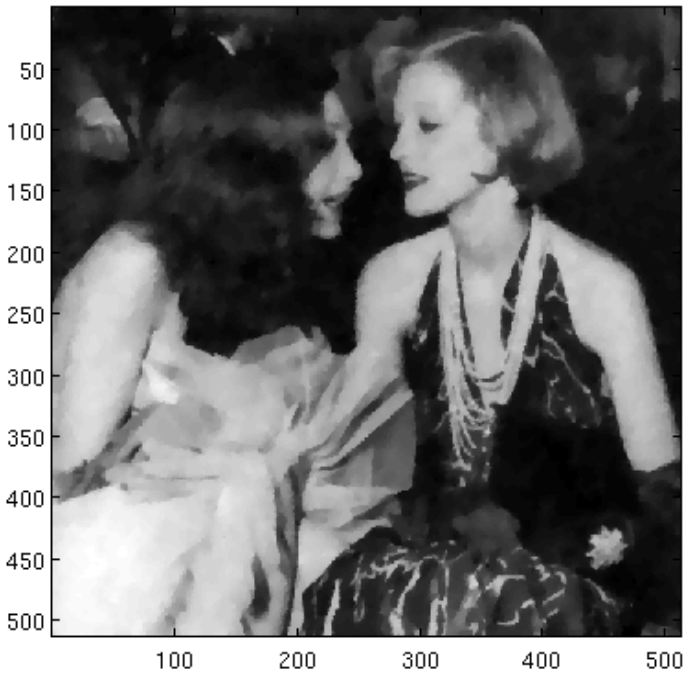
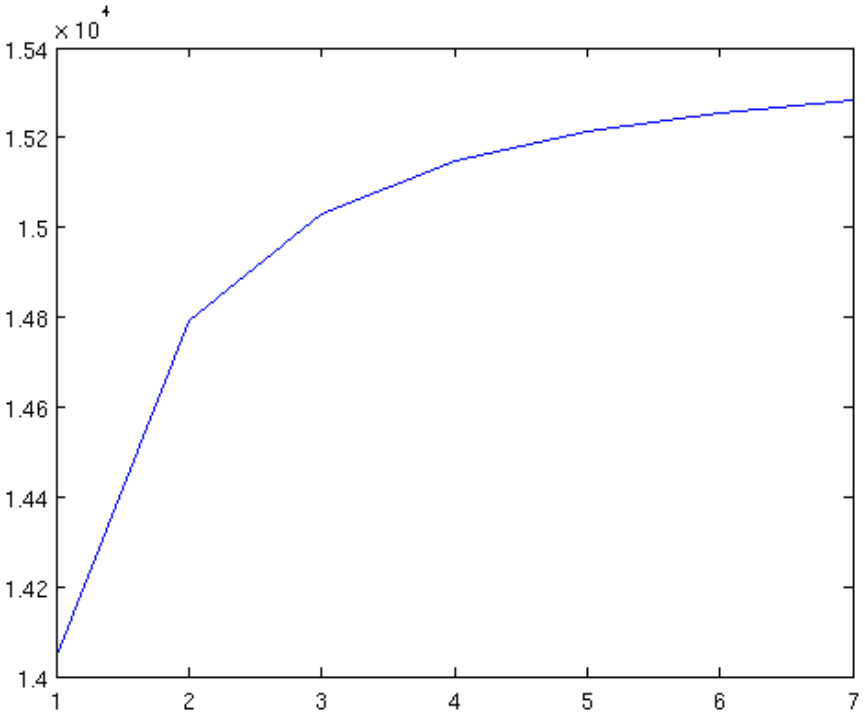


FIGURE 46 – Courbe de l'évolution de f_n en fonction du temps.



4.7 L'algorithme de Forward-Backward.

4.7.1 Cadre et présentation : le point de vue proximal.

Le cadre d'application de l'algorithme de Forward-Backward est de manière générale la minimisation de problème d'optimisation convexe de la forme :

$$J(v_0) = \min_{v \in \mathbb{R}^n} f_1(v) + f_2(v) + \cdots + f_r(v).$$

On traitera en exemple le cas où $r = 2$. L'algorithme de Backward-forward va consister à transformer le problème de minimisation de chaque fonctionnelle en projection de la condition initiale sur un sous espace convexe bien choisit. Si l'on suppose que chaque fonctionnelle f_i est la fonction indicatrice d'un ensemble convexe fermé C_i alors on peut donner un algorithme qui sera, si P_{C_i} est la projection sur C_i,

$$x_{n+1} = P_{C_1} \cdots PC_r(x_n).$$

Si l'on considère que l'on peut minimiser une fonction lisse par projection sur la sous-différentielle, on a la généralisation de l'algorithme de **gradient projeté**.

Également, si l'on se place dans un Banach de dimension finie, et que l'on choisit pour chaque C_i un hyperplan, on retrouve l'algorithme de **Kaczmarz** (ou ART).

Dans notre exemple, nous traiterons l'éternel problème de débruitage :

$$\min_{u \in X} \frac{||u - g||^2}{2\lambda} + J(u), \tag{5}$$

où $J(u)$ désigne la variation totale de notre image.

Ici on identifie nos deux fonctionnelles :

$$f_1(u) = \frac{||u - g||^2}{2\lambda} \text{ et}$$
$$f_2(u) = J(u).$$

La première est différentiable donc la projection P_{C_1} va consister en une simple descente de gradient. Quand à la projection P_{C_2}, l'algorithme de Chambolle nous donne une méthode de projection que l'on a notée $\pi_{\lambda K}$.

On a donc un algorithme, dit de **Forward-Backward**, donné par :

$$x_{n+1} \leftarrow \pi_{\lambda K} \left(x - \gamma_n \nabla f_1(x_n) \right).$$

4.7.2 Convergence.

Théorème 7
Soient $f_0 \in \Gamma_0$ et $f_1 : \mathbb{R}^N \to \mathbb{R}$ convexe et différentiable telle que :

$$\forall x, y \in \mathbb{R}^N \ , \ ||\nabla f_1(x) - \nabla f_1(y)|| \leqslant \beta ||x - y|| \ ,$$

où $\beta > 0$. On suppose aussi que $f_0 + f_1$ est coercive. Alors il est montré dans [2] que le problème

$$J(v_0) = \min_{v \in \mathbb{R}^n} f_0(v) + f_1(v)$$

admet une unique solution caractérisée par l'équation à point fixe

$$x = prox_{\lambda f_0} \left(x - \gamma \nabla f_1(x) \right).$$

4.7.3 Implémentation.

On utilisera ici la projection utilisée dans la section 4.6.2, que l'on avait appelée

```
function Im=chambolle_simple( N,lambda,g)
```

Notre algorithme de Forward-Backward sera écrit de la manière suivante :

```
for itt=1:N
   v=v-gamma*(v-g)/lambda;
   v=chambolle_simple(iterations_gradient,lambda,v);
end
```

En itérant l'algorithme, on obtient les résultats suivants :

FIGURE 47 – Image bruitée par un bruit additif gaussien d'écart-type 30.

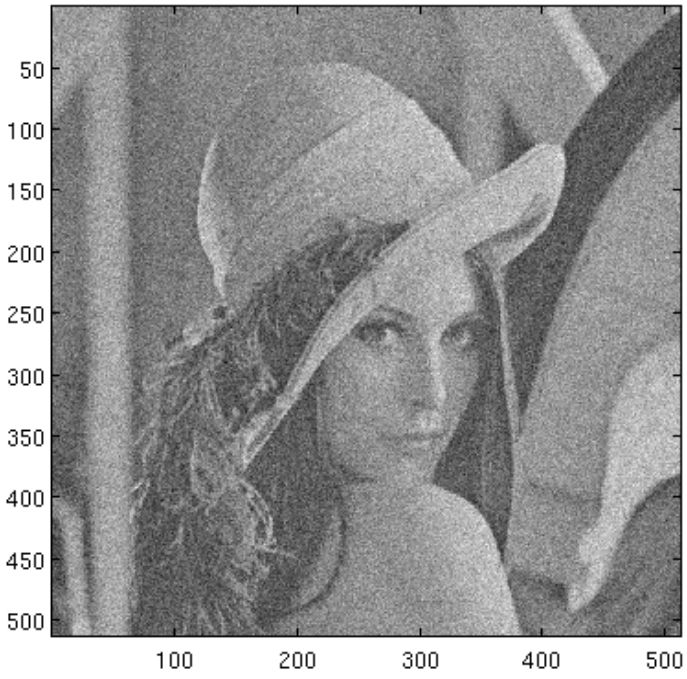
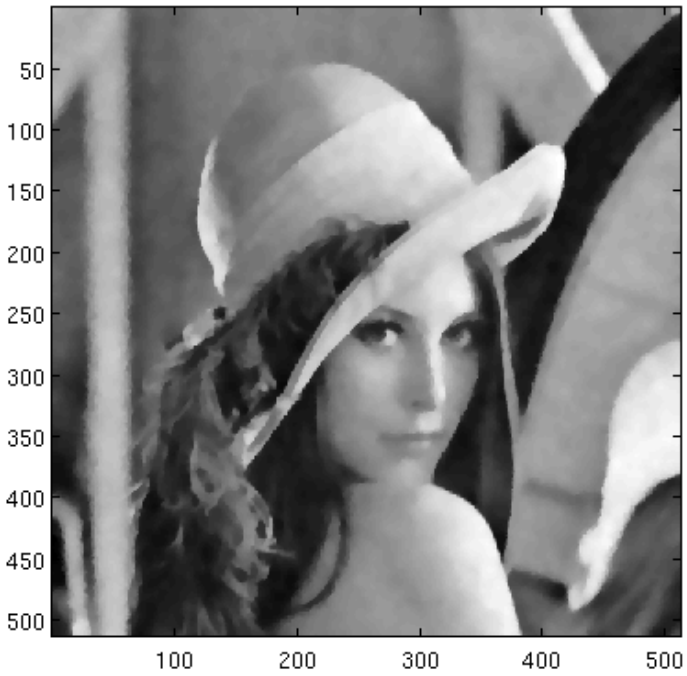


FIGURE 48 – Image débruitée par l'algorithme de Forward-Backward.



Ici, l'algorithme de Forward-Backward permet de minimiser le même critère que celui que l'on a minimisé par simple projection. Si l'on compare cette méthode à la projection simple, on voit qu'il y a une légère accélération de l'ordre de 2. Pour une norme-2 de l'erreur équivalente, on passe de 26 secondes pour la méthode par simple projection, contre 13 secondes pour la méthode de Forward-Backward.

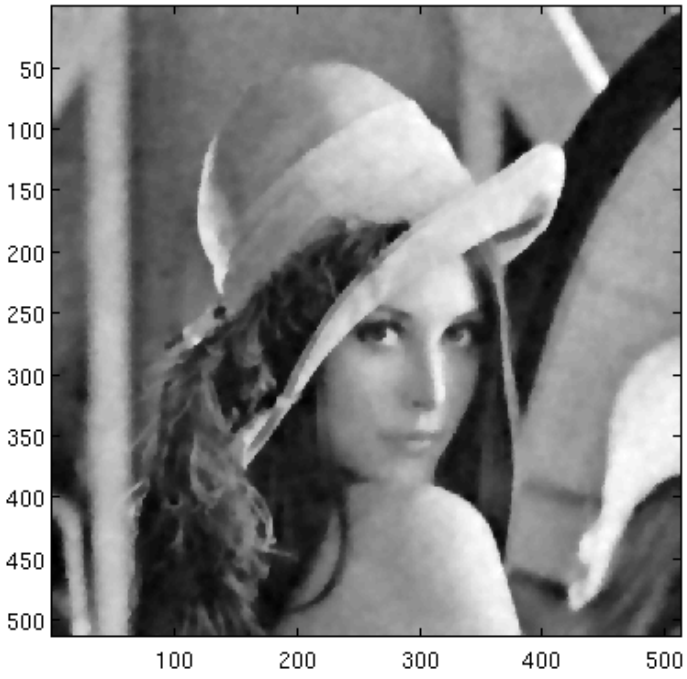
4.8 Accélération de Beck-Teboulle.

Un algorithme plus rapide encore, est proposé dans [5]. Il s'applique dans le cadre général du problème cadré par le théorème 7 : Supposons posés $x_0 \in \mathbb{R}^n$. Initialisons z_0 à x_0 et $t_0 = 1$ On itère ensuite :

- $y_n = z_n - \beta^{-1}\nabla f_2(z_n)$
- $x_{n+1} = \text{prox}_{\beta^{-1}f_1}(y_n)$
- $t_{n+1} = \dfrac{1+\sqrt{4t_n^2+1}}{2}$
- $\lambda_n = 1 + \dfrac{t_n - 1}{t_{n+1}}$
- $z_{n+1} = x_n + \lambda_n(x_{n+1} - x_n)$.

Les résultats sont alors similaires à une projection simple :

FIGURE 49 – Image débruitée par l'algorithme de Forward-Backward, accéléré par la méthode de Beck-Téboulle.



Mais il ne faut que 6 secondes pour exécuter l'algorithme, ce qui l'améliore encore.

4.9 Application à la déconvolution.

On veut déconvoluer en minimisant le critère :

$$\min_{u \in X} \frac{||T * u - g||^2}{2\lambda} + J(u). \tag{6}$$

On a donc un algorithme donné par Forward-Backward car le premier terme est dérivable :

$$x_{n+1} \leftarrow \pi_{\lambda K} \left(x - \gamma_n \nabla f_1(x_n) \right).$$

On détermine $\nabla f_1(x_n)$:

$$\begin{aligned} \nabla f_1(u) &= \nabla \left[||Tu - g||_2^2 \right] \\ &= \nabla \left[u^t T^t T u - 2 u^t T^t g + ||g||_2^2 \right]. \\ &= 2T^t T u - 2T^t g \end{aligned}$$

L'algorithme est donné par :

$$x_{n+1} \leftarrow \pi_{\lambda K} \left(x - \gamma_n (2T^t T x_n - 2T^t g) \right).$$

Cela donne les résultats suivants :

FIGURE 50 – Image convoluée (par un noyau gaussien) et bruitée.

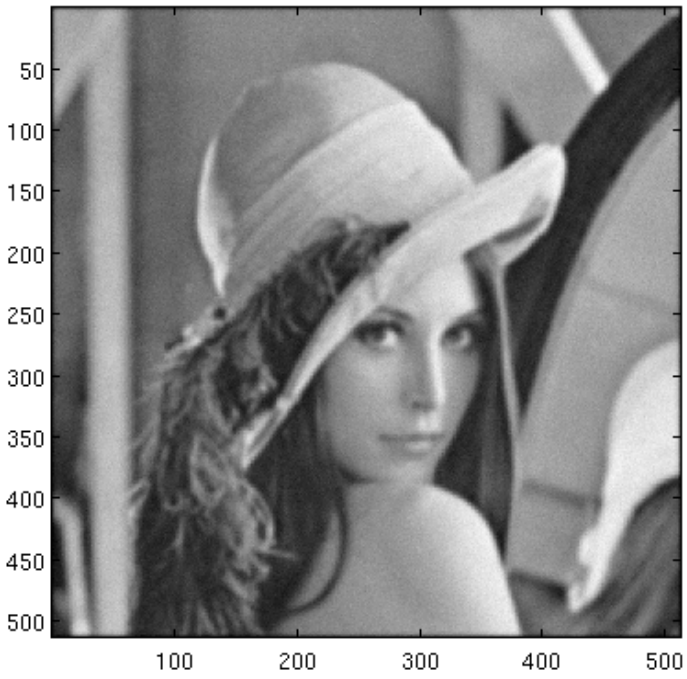


FIGURE 51 – Image déconvolée par l'algorithme de Forward-Backward, avec une régularisation TV.

On passe d'une erreur relative de 23.8533 à 0.9971. Ce sont des résultats équivalents à ceux obtenus avec l'utilisation de la norme $L^1 - L^2$, néanmoins la qualité visuelle est

nettement supérieure puisqu'on a supprimé l'effet de 'moiré'.

4.10 Application à l'inpainting.

On veut utiliser l'algorithme de Forward-Backward pour réaliser l'inpainting d'une image en cherchant à minimiser la *variation totale*. On choisit donc de le construire avec le formalisme des opérateurs proximaux. Pour réaliser cela on explicite les différentes projections qui vont intervenir. Il y a ici, en l'occurrence, 2. La première projection est celle qui minimise la variation totale de l'image, donnée par Chambolle dans [1], et que l'on a noté jusqu'à présent $\pi_{\lambda K}$. La deuxième projection sera celle qui minimise la distance entre l'image masquée et l'image courante de l'algorithme. On donne ainsi :

$$P_2(v_{i,j}) = \left\{ \begin{array}{c} v_{i,j} \text{ si le masque est à valeur } 0 \\ g_{i,j} \text{ sinon} \end{array} \right\}.$$

C'est une projection sur un sous-espace vectoriel (par conséquent, un ensemble fermé et convexe) d'un espace hilbertien. L'algorithme de Forward-Backward deviendra alors :

$$x_{n+1} \leftarrow \pi_{\lambda K}\left(P_2(x_n)\right).$$

On peut aussi augmenter l'efficacité de l'algorithme en lui insérant la méthode de Beck-Téboulle :

- $y_n = z_n + \dfrac{1}{\beta\lambda} M(g - z_n)$
- $x_{n+1} = \text{prox}_{\beta^{-1} f_1}(y_n)$
- $t_{n+1} = \dfrac{1 + \sqrt{4t_n^2 + 1}}{2}$
- $\lambda_n = 1 + \dfrac{t_n - 1}{t_{n+1}}$
- $z_{n+1} = x_n + \lambda_n(x_{n+1} - x_n)$,

avec g qui est l'image masquée par le masque M.

On obtient alors le résultat suivant :

FIGURE 52 – Image initiale : 50% des données sont manquantes.

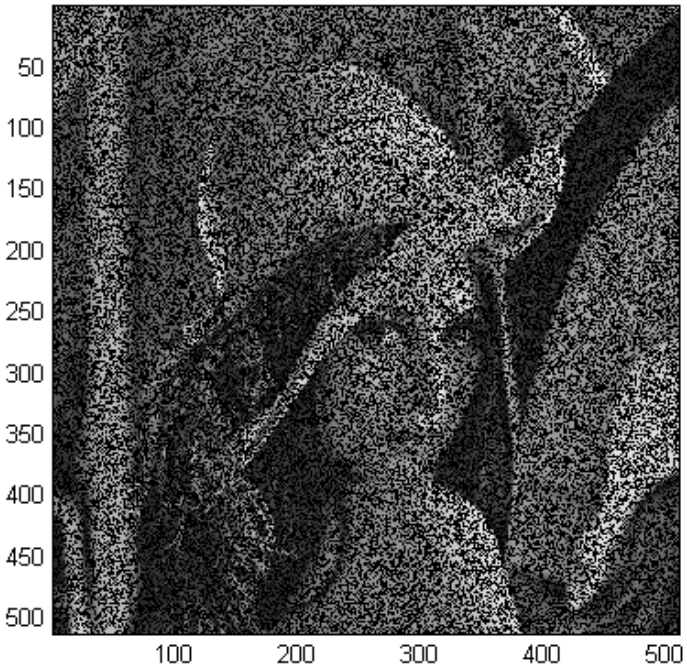
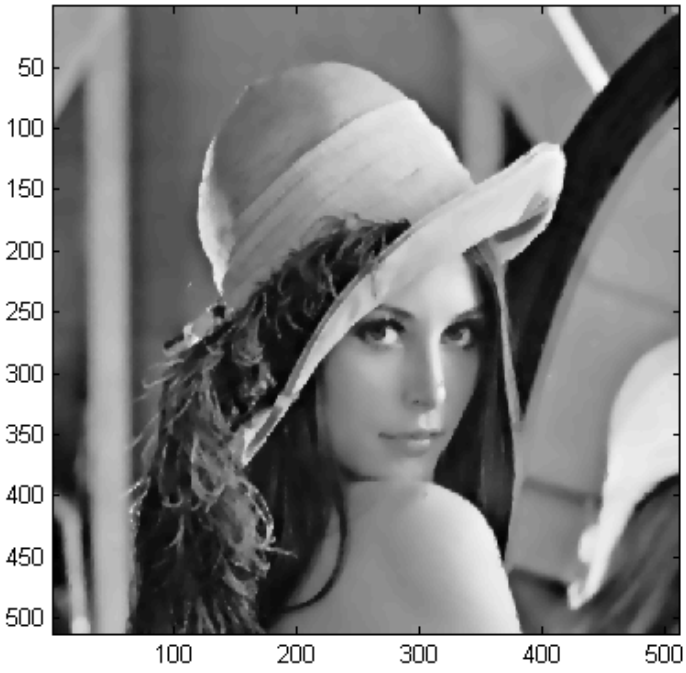


FIGURE 53 – Image inpaintée par l'algorithme de Forward-Backward amélioré par la méthode de Beck-Téboulle.



On remarque encore ici que la méthode de Beck-Téboulle accélère l'algorithme, ici, du simple au double.

5 Utilisation d'algorithmes "primal-dual" pour des problèmes non lisses.

5.1 Le critère TV-L^1 . Résolution par algorithme primal-dual.

5.1.1 Mise en place théorique de l'algorithme primal-dual.

On veut maintenant minimiser le critère :

$$\min_{u\in X} \lambda||u-g||_1 + J(u), \text{ avec :}$$

$$J(u) = \sum_{1\leqslant i,j\leqslant N} \sqrt{(\nabla u^1_{i,j})^2 + (\nabla u^2_{i,j})^2}.$$

On pose le problème dual associé [9] :

$$\min_{u\in X}\max_{p\in Y} \left[-\langle u|\text{div}p\rangle_X + \lambda||u-g||_1 - \delta_P(p)\right] \ , \tag{7}$$

avec

$$\delta_P(p) = \begin{cases} 0 \text{ si } p\in P \ , \\ +\infty \text{ sinon.} \end{cases}$$

Et $P = \{p\in Y : ||p||_\infty \leqslant 1\}$.

Ce problème est d'une forme que l'on identifie aisément à la forme généralisée :

$$\min_{u\in X}\max_{p\in Y} \left[\langle Ku|p\rangle_X + G(u) - F^*(y)\right] \ , \tag{8}$$

9. Voir pour cela [4] ou [11] page 2.

associé au problème dit "primal" :

$$\min_{u\in X}\left[F(Kx)+G(x)\right]\ ,$$

où $G : X \leftarrow [0,+\infty]$; $F^* : Y \leftarrow [0,+\infty]$ est propre, semi-continue inférieurement, et convexe ; K est un opérateur linéaire continu.

Un algorithme de résolution de base, est donné dans [11] :

Algorithme : 1

- On initialise τ, $\sigma > 0$, $\theta \in [0,1]$, $(x^0, y^0) \in X \times Y$ et $\overline{x}^0 = x^0$.
- On itère :
  - $y^{n+1} = (I + \sigma\partial F^*)^{-1}(y^n + \sigma K\overline{x}^n)$
  - $x^{n+1} = (I + \tau\partial G)^{-1}(x^n + \tau K^* y^{n+1})$
  - $\overline{x}^n = x^{n+1} + \theta(x^{n+1} - x^n)$.

Ce que l'on peut écrire de la façon suivante :

On itère :

- $y^{n+1} = \text{prox}_{\sigma F}(y^n + \sigma K\overline{x}^n)$
- $x^{n+1} = \text{prox}_{\tau G}(x^n + \tau K^* y^{n+1})$
- $\overline{x}^n = x^{n+1} + \theta(x^{n+1} - x^n)$.

Identifions dans 7 les éléments de 8 afin de pouvoir appliquer l'algorithme 1 :

$$Kx = \nabla(x)\ ,$$

$$F^* = \delta_P(p)\ ,$$

$$G = \lambda||u-g||_1.$$

Cherchons maintenant à expliciter $\text{prox}_{\tau G} = (I + \tau\partial G)^{-1}$ et $\text{prox}_{\sigma F} = (I + \sigma\partial F^*)^{-1}$.

Pour $(I + \sigma\partial F^*)^{-1}$, on se réfère au théorème 5. Il suffit de donner la projection sur P. Si $p = (p_{i,j})_{1\leqslant i,j\leqslant N}$ alors

$$\text{prox}_{\sigma F}(p) = (I + \sigma\partial F^*)^{-1}(p) = \begin{cases} p_{i,j} & \text{si } ||p_{i,j}||_\infty \leqslant 1 \\ \dfrac{p_{i,j}}{||p_{i,j}||_\infty} & \text{sinon.} \end{cases}$$

Pour $(I + \tau\partial G)^{-1}$, on se réfère cette fois ci au théorème 4.

Proposition 4

On a :

$$u = (I + \tau\partial G)^{-1}(\tilde{u}) \Leftrightarrow u_{i,j} = \begin{cases} \tilde{u}_{i,j} - \tau\lambda & si\ \tilde{u}_{i,j} - g_{i,j} > \tau\lambda \\ \tilde{u}_{i,j} + \tau\lambda & si\ \tilde{u}_{i,j} - g_{i,j} < -\tau\lambda \\ g_{i,j} & si\ |\tilde{u}_{i,j} - g_{i,j}| \leqslant \tau\lambda \end{cases}.$$

Preuve Par définition :

$$u = (I + \tau\partial G)^{-1}(\tilde{u}) \Leftrightarrow u \text{ minimise } \frac{||u-\tilde{u}||_2^2}{2} + \tau\lambda||u-g||_1.$$

Ce qui s'écrit coordonnée par coordonnée :

$$\frac{(u_{i,j}-\tilde{u}_{i,j})^2}{2} + \tau\lambda|u_{i,j} - g_{i,j}|.$$

On minimise cela par dérivation en distinguant quelques cas ; les équations de Legendre sont :

$$\frac{\partial}{\partial u_{i,j}} = \begin{cases} u_{i,j} - \tilde{u}_{i,j} + \tau\lambda & \text{si } u_{i,j} - g_{i,j} > 0 \\ u_{i,j} - \tilde{u}_{i,j} - \tau\lambda & \text{si } u_{i,j} - g_{i,j} < 0 \\ u_{i,j} - \tilde{u}_{i,j} & \text{si } u_{i,j} - g_{i,j} = 0. \end{cases}$$

Les équations de Legendre prennent alors la forme suivante :

$$\begin{cases} u_{i,j} - \tilde{u}_{i,j} + \tau\lambda = 0 & \text{si } u_{i,j} - g_{i,j} > 0 \\ u_{i,j} - \tilde{u}_{i,j} - \tau\lambda = 0 & \text{si } u_{i,j} - g_{i,j} < 0 \\ u_{i,j} - \tilde{u}_{i,j} = 0 & \text{si } u_{i,j} - g_{i,j} = 0. \end{cases}$$

Ou encore :

$$\begin{cases} u_{i,j} - \tilde{u}_{i,j} + \tau\lambda = 0 & \text{si } u_{i,j} - g_{i,j} > 0 \\ u_{i,j} - \tilde{u}_{i,j} - \tau\lambda = 0 & \text{si } u_{i,j} - g_{i,j} < 0 \\ u_{i,j} - \tilde{u}_{i,j} = 0 & \text{si } u_{i,j} - g_{i,j} = 0. \end{cases}$$

Ainsi le minimiseur de $u_{i,j}$ est donné par :

$$\begin{cases} \tilde{u}_{i,j} - \tau\lambda & \text{si } \tilde{u}_{i,j} - \tau\lambda - g_{i,j} > 0 \\ \tilde{u}_{i,j} + \tau\lambda & \text{si } \tilde{u}_{i,j} + \tau\lambda - g_{i,j} < 0 \\ \tilde{u}_{i,j} & \text{si } u_{i,j} - g_{i,j} = 0. \end{cases}$$

i.e.

$$\begin{cases} \tilde{u}_{i,j} - \tau\lambda & \text{si } \tilde{u}_{i,j} - \tau\lambda - g_{i,j} > 0 \\ \tilde{u}_{i,j} + \tau\lambda & \text{si } \tilde{u}_{i,j} + \tau\lambda - g_{i,j} < 0 \\ \tilde{u}_{i,j} & \text{si } |\tilde{u}_{i,j} - g_{i,j}| \leqslant \tau\lambda. \end{cases}$$

5.1.2 Application sur un bruit "poivre et sel".

Cela s'implémente aisément et nous donne les résultats visibles sur les figures ci-dessous. Ici, nous n'avons pas pris un bruit gaussien centré réduit comme d'habitude mais un bruit dit "poivre et sel" qui est un bruit gaussien de forte variance qui s'applique sur les pixels pris au hasard sur l'image (en l'occurrence un bruit d'écart-type 90 sur 25% des pixels.)

FIGURE 54 – Image bruitée "sel et poivre".

FIGURE 55 – Image débruitée par minimisation d'un critère TV-L^1.

FIGURE 56 – Image débruitée par minimisation de TV (algorithme de Chambolle amélioré.)

On peut comparer la qualité de ces résultats avec ceux obtenus avec la norme-2 seule :

	Algorithme de Chambolle amélioré.	Minimisation d'un critère TV-L^1
SNR avant débruitage	5.2669	5.2527
SNR après débruitage	8.9847	9.7420

.

FIGURE 57 – Image bruitée "sel et poivre" sur 80% des pixels.

FIGURE 58 – Image débruitée par minimisation d'un critère TV-L^1.

On remarque donc que l'algorithme $TV - L^1$ est très efficace sur le bruit de type "poivre et sel". En revanche il est complètement inefficace sur le bruit additif gaussien. Ici, on a cherché les meilleurs paramètre possible pour débruiter :

FIGURE 59 – Image bruitée par un bruit additif gaussien d'écart type 30.

FIGURE 60 – Le meilleur débruitage obtenu par minimisation du critère $TV - L^1$.

5.2 Inpainting par minimisation de la variation totale : approche par algorithme primal-dual.

On définit le modèle

$$\min_{u\in X} J(u) + \frac{\lambda}{2}\sum_{i,j\in D\setminus I}(u_{i,j}-g_{i,j})^2 \ ,$$

où I est le domaine des données perdues (domaine d'inpainting [10])

On réécrit le problème dual :

$$\min_{u\in v}\max_{p\in Y}\langle\nabla u|p\rangle + \frac{\lambda}{2}\sum_{i,j\in D\setminus I}(u_{i,j}-g_{i,j})^2 + \delta_P(p).$$

De même qu'avant, identifions dans 7 les éléments de 8 afin de pouvoir appliquer l'algorithme 1 :

$$Kx = \nabla(x) \ ,$$
$$F^* = \delta_P(p) \ ,$$
$$G = \frac{\lambda}{2}\sum_{i,j\in D\setminus I}(u_{i,j}-g_{i,j})^2.$$

Proposition 5
On a :

$$u = (I+\tau\partial G)^{-1}(\tilde{u}) \Leftrightarrow u_{i,j} = \begin{cases} \tilde{u}_{i,j} & si\ (i,j)\in I \\ \dfrac{\tilde{u}_{i,j}+\tau\lambda g_{i,j}}{1+\tau\lambda} & si\ (i,j)\in D\setminus I \end{cases} .$$

Preuve

$$u = (I+\tau\partial G)^{-1}(\tilde{u}) \Leftrightarrow u_{i,j} \text{ minimise } \begin{cases} \dfrac{(u_{i,j}-\tilde{u}_{i,j})^2}{2} + \dfrac{\tau\lambda}{2}(u_{i,j}-g_{i,j})^2 & \text{si } (i,j)\in D\setminus I \\ \dfrac{(u_{i,j}-\tilde{u}_{i,j})^2}{2} & \text{si } (i,j)\in I. \end{cases}$$

On dérive donc par rapport à $u_{i,j}$ pour minimiser :

$$\frac{\partial}{\partial u_{i,j}} = \begin{cases} u_{i,j}-\tilde{u}_{i,j}+\tau\lambda u_{i,j}-g_{i,j} & \text{si } (i,j)\in D\setminus I \\ u_{i,j}-\tilde{u}_{i,j} & \text{si } (i,j)\in I. \end{cases}$$

L'équation de Legendre devient :

$$\begin{cases} u_{i,j}-\tilde{u}_{i,j}+\tau\lambda u_{i,j}-g_{i,j}=0 & \text{si } (i,j)\in D\setminus I \\ u_{i,j}-\tilde{u}_{i,j}=0 & \text{si } (i,j)\in I. \end{cases}$$

Donc le minimum est atteint pour :

$$u_{i,j} = \begin{cases} \tilde{u}_{i,j} & \text{si } (i,j)\in I \\ \dfrac{\tilde{u}_{i,j}+\tau\lambda g_{i,j}}{1+\tau\lambda} & \text{si } (i,j)\in D\setminus I \end{cases} .$$

Cela s'implémente facilement quand on on déjà implémenté $TV-L^1$. Et cela donne :

10. Les "trous à boucher".

FIGURE 61 – Image masquée à 80%.

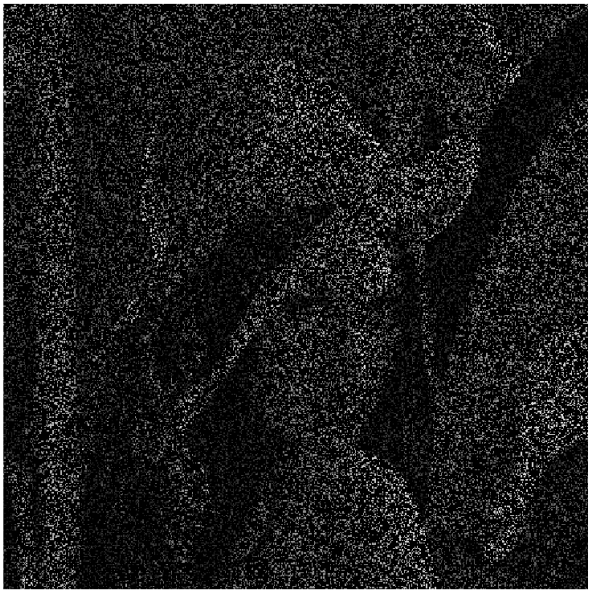


FIGURE 62 – Image inpaintée par algorithme dual de minimisation TV.

FIGURE 63 – Qui se cache derrière ce masque à 90% nul ?

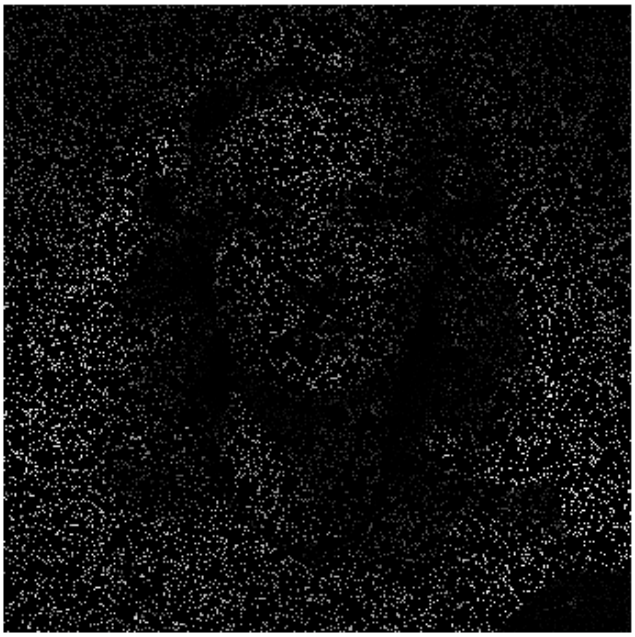


FIGURE 64 – Michèle Morgan !

On commence à voir ici que l'algorithme atteint ses limites, l'image est dégradée. Si on pousse à 98% de données manquantes, le résultat devient vraiment dégradé. (alors qu'à 97%, il est encore possible de reconnaitre un visage !)

FIGURE 65 – Image inpaintée par algorithme dual de minimisation TV.

FIGURE 66 – Limites de l'inpainting par minimisation de TV.

Ici, on voit que l'algorithme ne parvient pas à retrouver les données manquantes, ce qui est normal puisqu'il manque de l'information.

L'implémentation en divers langages de programmation (voir code en annexe) m'a

permis d'apprécier les différences de puissance entre le logiciel Matlab, la machine virtuelle java, et une implémentation en C, avec le mode '-03' [11] d'optimisation de gcc . Ci-dessous les divers temps d'exécution de 6000 itérations de l'algorithme précédent, sur AMD Sempron 140 Processor (mono-cœur), 2.70 GHz, 2.00 Go de RAM. :

Langage	Temps d'exécution (s.)
Matlab R2011a	962
Java 1.7	388
C	315

11. L'option '-03' de gcc permet d'optimiser la compilation pour plus de rapidité d'exécution du programme. Par exemple, sans cette option l'exécution de ce programme dure environ 410 secondes.

Table des figures

Références

[1] Antonin Chambolle *An algorithme for Total Variation Minimisation and Application*,

[2] Combettes, Wajs, *Signal recovery by proximal forward-backward splitting. Multiscale Model.*, 1168-1200 (2005).

[3] Jean Alexandre Dieudonné, *Calcul infinitésimal.*, Hermann, 1968.

[4] PG Ciarlet, *Introduction à l'analyse numérique matricielle et à l'optimisation.*, Dunod, 2006.

[5] Combettes, Pesquet, *Proximal Splitting Methods in Signal Processing.*

[6] Jean Alexandre Dieudonné, *Les fondements de l'analyse moderne, Tome 1*, Gauthier-Vilars

[7] Alain Yger, Jaques-Arthur Weil, *Mathématiques appliquées L3*, Pearson Éducation, 2009.rs, 1972.

[8] Jean-Baptiste Hiriart-Urruty, *Optimisation et analyse convexe*, PUF, 1988.

[9] Jean-Baptiste Hiriart-Urruty et Claude Lemaréchal, *Convex Analysis and Minimization algorithms 1.*, Springer-Verlag, 1993.

[10] Jean-Baptiste Hiriart-Urruty et Claude Lemaréchal, *Convex Analysis and Minimization algorithms 2.*, Springer-Verlag, 1993.

[11] Antonin Chambolle, Thomas Pock, *A first-order primal-dual algorithm for convex problems with applications to imaging*, hal-00490826-v1, 9/06/2010.

[12] Claude Delannoy, *Langage C*, Eyrolles, 2005.

[13] Rogers Cadenhead, Laura Lemay, *Java 2*, CampusPress, 2004.

[14] Jean-François Aujol, *Introduction to optimization*, 23 March 2009.

Annexe : code Matlab.

Algorithmes et codes pour les méthodes sans contrainte.

```
%% Dessin de la fonction a minimiser.
figure
[X,Y]=meshgrid(-4:0.3:4,-5:0.3:12);
Z=(X-1).^2+10*(X.^2-Y).^2;
hold on
plot3(1,1,(1-1).^2+10*(1.^2-1).^2+1,'--rs','LineWidth',1,...
                    'MarkerEdgeColor','k',...
                    'MarkerFaceColor','g',...
                    'MarkerSize',8)
surf(X,Y,Z,'EdgeColor','black')

%%
[U,V]=eig(HessJ([1,1]))

%% Methode de newton.
clc
close all;

tic
S1=NewtonJ([-1 1],3);
toc
% Elapsed time is 0.004419 seconds.

plot(S1(1,:),S1(2,:),'x');
axis([-1.5 1.5 -3.5 1.5])
figure
hold on
plot3(S1(1,:),S1(2,:),Jcalc(S1),'xr');
surf(X,Y,Z,'EdgeColor','black')
%title('Iteration de la methode de Newton menant a solution.');

%% Methode de quasi-Newton.
clc
close all;

tic
S2=QuasiNewtonJ([-1 1],1500,20,0.06);
toc
% Elapsed time is 0.074026 seconds.

plot(S2(1,:),S2(2,:),'x');
figure
hold on
plot3(S2(1,:),S2(2,:),Jcalc(S2),'xr');
surf(X,Y,Z,'EdgeColor','black')
%title('Iteration de la methode de quasi Newton menant a solution.');

%% Gradient a pas constant.
clc
close all;

tic
S3=MethodeGradientJ([-1 1],1000,0.04);
toc
```

```
% Elapsed time is 0.075924 seconds.

plot(S3(1,:),S3(2,:),'x');
figure
hold on
plot3(S3(1,:),S3(2,:),Jcalc(S3),'xr');
surf(X,Y,Z,'EdgeColor','black')
%title('Iteration de la methode de gradient a  pas fixe menant a solution.');

%% Methode de relaxation.
clc
close all;

tic
S4=MethodeRelaxationJ([-2,-2],1000,10^(-15),0.001);
toc
% Elapsed time is 0.151317 seconds.

figure
hold on
plot(S4(1,:),S4(2,:),'x');
figure
hold on
plot3(S4(1,:),S4(2,:),Jcalc(S4),'xr');
surf(X,Y,Z,'EdgeColor','black')

%% Methode de gradient a  pas optimal.
clc
close all;

tic
S5=MethodeGradientPasOptimalJ([-1,1],200,10^(-15),0.001);
toc
% Elapsed time is 3.546075 seconds.

figure
hold on
plot(S5(1,:),S5(2,:),'x');
figure
hold on
plot3(S5(1,:),S5(2,:),Jcalc(S5),'xr');
surf(X,Y,Z,'EdgeColor','black')

%% Methode de gradient conjugue.
clc
close all;

tic
S6=MethodeGradientConjugue([-1,1],15,10^(-10),0.0000001);
toc
% Elapsed time is 6.903303 seconds.

figure
hold on
plot(S6(1,:),S6(2,:),'x');
figure
hold on
plot3(S6(1,:),S6(2,:),Jcalc(S6),'xr');
```

```
surf(X,Y,Z,'EdgeColor','black')

%%
figure
hold on
plot(S2(1,:),S2(2,:),'xr');
plot(S3(1,:),S3(2,:),'xg');
plot(S4(1,:),S4(2,:),'xy');
plot(S5(1,:),S5(2,:),'xc');
plot(S1(1,:),S1(2,:),'x');
plot(S6(1,:),S6(2,:),'xm');
%% Debut des contraintes.
clc
close all

[m,S,B]=dichotomieXJ2([0  1],-3,-10,10,0.00001)

hold on
plot(S(1,:),'o')
plot(S(2,:),'xr')

figure
hold on
plot3(B(1,:),B(2,:),Jcalc(B),'xr');
surf(X,Y,Z,'EdgeColor','black')

figure
plot(B(1,:),B(2,:),'x')

%% Minimum sur un carre.
clc
close all
C=[-4,-1,6,10];
P= RelaxationSurCarre(C(1),C(2),C(3),C(4),[-2,8],15,0.01)

figure
hold on
plot(P(1,:),P(2,:),'x')
CD=[[C(1);C(3)],[C(1);C(4)],[C(2);C(4)],[C(2);C(3)],[C(1);C(3)]];
plot(CD(1,:),CD(2,:),'r')
axis([-4.5,-0.5,5.5,10.5])
% figure
% hold on
% CD=[[C(1);C(3)],[C(1);C(4)],[C(2);C(4)],[C(2);C(3)]];
% plot3(CD(1,:),CD(2,:),Jcalc(CD),'r');
% plot3(P(1,:),P(2,:),Jcalc(P),'xr');
% surf(X,Y,Z,'EdgeColor','black')

%% Minimum sur un carre.
clc
close all
C=[-2,2,-2,2];
P= RelaxationSurCarre(C(1),C(2),C(3),C(4),[-1,1],15,0.000001)

figure
hold on
plot(P(1,:),P(2,:),'x')
```

```
CD=[[C(1);C(3)],[C(1);C(4)],[C(2);C(4)],[C(2);C(3)]];
plot(CD(1,:),CD(2,:),'r');
% plot3(P(1,:),P(2,:),Jcalc(P),'xr');
% surf(X,Y,Z,'EdgeColor','black')

%%
close all
T=0:0.1:6;
u=[5];
for i=1:4
    u=[u,u(end)-(u(end)^2-2)/(2*u(end))];
end
u1=kron(u,[1,1]);
v=kron(u.^2-2,[0,1]);
hold on
plot(T,T.^2-2)
plot([0,6],[0,0],'g')
plot(u1,v,'r')

function H = HessJ( X )
% Calcule le Hessien de la fonction J au point X.

H=[2+120*X(1)^2-40*X(2) ,-40*X(1) ;
    -40*X(1)  ,20];

end

function Y = gradJ( X)
% calcule le gradient de la fonction J au point x.

Y=[2*X(1)-2+40*X(1)^3-40*X(2)*X(1);-20*(X(1)^2-X(2))];

end

function S = NewtonJ( Init,N )
% NewtonJ calcule le minimum de la banane de Rosenbrock en
% partant de Init en N iterations.

Init=Init(:);
S=zeros(length(Init),N);
S(:,1)=Init;

for itt=2:N
    X=S(:,itt-1);
    H=HessJ(X);
    G=gradJ(X);
    Y=X-H^(-1)*G;
    S(:,itt)=Y;

end

end

function S = QuasiNewtonJ( Init,N, duree,param)
% QuasiNewtonJ calcule le minimum de la banane de Rosenbrock en
% partant de Init en N it rations.
```

```
Init=Init(:);
S=zeros(length(Init),N);
S(:,1)=Init;
c=0;
X=S(:,1);
H=HessJ(X);
InvH=H^(-1);

for itt=2:N
    c=c+1;
    X=S(:,itt-1);

    if c==duree
        H=HessJ(X);
        InvH=H^(-1);
        c=0;
    end

    G=gradJ(X);
    Y=X-param*InvH*G;
    S(:,itt)=Y;

end

end

function S = MethodeGradientJ( Init,N,param)
% MethodeGradientJ calcule le minimum de la banane de Rosenbrock en
% partant de Init en N iterations.

Init=Init(:);
S=zeros(length(Init),N);
S(:,1)=Init;

for itt=2:N
    X=S(:,itt-1);
    G=gradJ(X);
    Y=X-param/2*G;
    S(:,itt)=Y;
end

end

function S = QuasiNewtonJ( Init,N, duree,param)
% QuasiNewtonJ calcule le minimum de la banane de Rosenbrock en
% partant de Init en N it rations.

Init=Init(:);
S=zeros(length(Init),N);
S(:,1)=Init;
c=0;
X=S(:,1);
H=HessJ(X);
InvH=H^(-1);

for itt=2:N
    c=c+1;
```

```
    X=S(:,itt-1);

    if c==duree
        H=HessJ(X);
        InvH=H^(-1);
        c=0;
    end

    G=gradJ(X);
    Y=X-param*InvH*G;
    S(:,itt)=Y;

end

end

function S = MethodeRelaxationJ( Init,N,param1, param2)
% MethodeRelaxationJ calcule le minimum de la banane de Rosenbrock en
% partant de Init en N it rations.

Init=Init(:);
S=zeros(length(Init),N);
S(:,1)=Init;

for itt=2:N
    % r cup ration du dernier x.
    X=S(:,itt-1);

    % minimisation de J par rapport    x
    m=0;
    e=1;
    while e>param1
        l=m;
        m=l-param2*(2*l-2+40*l^3-40*X(2)*l);
        e=abs(m-l);
    end
    S(1,itt)=m;

    % minimisation de J par rapport    y.
    S(2,itt)=S(1,itt)^2;
end

end

function S = MethodeGradientPasOptimalJ( Init,N,param1,param2)
% MethodeGradientPasOptimalJ calcule le minimum de la banane de Rosenbrock en
% partant de Init en N iterations.

Init=Init(:);
S=zeros(length(Init),N);
S(:,1)=Init;

for itt=2:N
    X=S(:,itt-1);
    G=gradJ(X);

    % pour simplifier le codage
    dx=G(1);
```

```
    dy=G(2);
    x=X(1);
    y=X(2);

    % choix du pas optimal
    m=0;
    e=1;
    while e>param1
        l=m;
        m=l-param2*((2*(x+l*dx-1))*dx+(20*((x+l*dx)^2-y-l*dy))*((2*(x+l*dx))*dx-dy));
        e=abs(m-l);
    end

    Y=X+m*G;
    S(:,itt)=Y;
end
end

function S = MethodeGradientConjugue( Init,N,param1,param2)
% MethodeGradientConjugue calcule le minimum de la banane de Rosenbrock en
% partant de Init en N it rations.

Init=Init(:);
S=zeros(length(Init),N);
S(:,1)=Init;

% direction initiale.
d=gradJ(Init);

for itt=2:N
    X=S(:,itt-1);

    % pour simplifier le codage
    dx=d(1);
    dy=d(2);
    x=X(1);
    y=X(2);

    % choix du pas optimal
    m=0;
    e=1;
    while e>param1
        l=m;
        m=l-param2*((2*(x+l*dx-1))*dx+(20*((x+l*dx)^2-y-l*dy))*((2*(x+l*dx))*dx-dy));
        e=abs(m-l);
    end

    % iteration a proprement parler
    S(:,itt)=X+m*d;

    % choix de la direction
    Vul=gradJ(S(:,itt));
    Vul_1=gradJ(S(:,itt-1));
    d=Vul+((Vul'*(Vul-Vul_1))/(Vul_1'*Vul_1))*d;

end
end
```

```
function [born1,S,B] = dichotomieXJ2( Direction,c,a,b,err)
% dichotomieXJ2 retourne le minmum ( en position) de la fonctionnelle J sur la droite
% d finie par Direction et c et compris entre a et b evec une erreur
% inf rieure en valeur absolue    err. S et B sont les it rations us e par
% l'algorithme.

S=[];
B=[];
D1=Direction(1);
D2=Direction(2);
born1=a;
born2=b;

while abs(born1-born2)>err
    b=(born1+born2)/2;
    m=Jcalc([D1*b+c*D2;D2*b+c*D1]);

    if m>Jcalc([D1*born1+c*D2;D2*born1+c*D1]);
        born2=b;
    else
        born1=b;
    end

    B=[B,[D1*b+c*D2;D2*b+c*D1]];
    S=[S,[born1;born2]];

end

end
```

Algorithmes et codes pour les méthodes avec contraintes.

```
function S = RelaxationSurCarre( Xmin, Xmax, Ymin, Ymax, Init,N, param )
% RelaxationSurCarre retourne le minimum de la banane de rosenbach sur le
%carre defini par les 4 premiers arguments.

Init=Init(:);
S=zeros(length(Init),N);
S(:,1)=Init;

for itt=2:N
    % recuperation du dernier x.
    X=S(:,itt-1);

    % minimisation de J par rapport a x
    X=dichotomieXJ3([1,0],X(2),Xmin,Xmax,param);

    % minimisation de J par rapport a y.
    X=dichotomieXJ3([0,1],X(1),Ymin,Ymax,param);

    S(:,itt)=X;
end

end

%% Gradient projete.
clc
A=[1,2;-1,3];
```

```
b=[-3;4];

U=GradProj(A,b,0.01,40,'ProjPositif');
v=A^-1*b
figure
hold on
plot(U(1,:),U(2,:),'x')
plot(v(1),v(2),'go')
plot(0,0,'ro')
axis([-3.5,0.5,-0.5,4.5])


%% Gradient projete sur cylindre.

clc
clear all
close all

A=[2 1 0
1 3 1
1 0 2];
b=[3
    1
    3];

u=GradProj(A,b,0.01,150,'ProjCyl');

hold on
plot3(u(1,:),u(2,:),u(3,:),'--rs','LineWidth',1,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',8)
[X,Y,Z] = cylinder(1,20);
mesh(X,Y,4*Z)
%surf(X,Y,4*Z)


%% Penalisation.
clc, clear all, close all

S=gradContraint([-1;-1],2000,1000,50);

figure
hold on
plot(S(1,:),S(2,:))
plot([1,3],[1,-1],'r')
plot([0,1],[3/2,1],'g')

S=gradContraint([-1;1],2000,1000,50);

figure
hold on
plot(S(1,:),S(2,:))
plot([1,3],[1,-1],'r')
plot([0,1],[3/2,1],'g')

%% Dessin de la fonction a minimiser.
```

```
[X,Y]=meshgrid(-40:3:40,-50:3:120);
Z=X.^2+Y.^2-14*X-6*Y-7;

surf(X,Y,Z,'EdgeColor','black')

%% dichotomie
close all
N=30
A=zeros(1,N);
B=[5,zeros(1,N-1)];


for i=2:N
    c=A(i-1)+1/3*(B(i-1)-A(i-1));
    d=A(i-1)+2/3*(B(i-1)-A(i-1));
    f_c=(c-2)^2;
    f_d=(d-2)^2;
    if f_c>f_d
        A(i)=c;
        B(i)=B(i-1);
    else
        B(i)=d;
        A(i)=A(i-1);
    end
end

figure
hold on
plot(A)
plot(B,'r')

function U = GradProj( A,b,tau,r ,ProjOp)
% Gradient projete minimise la fonctionnelle J(x)=||b-Ax||^2 pour x>=0.

P=str2func(ProjOp);
u=b(:);
U=zeros(length(b),1);
U(:,1)=b;

for itt=1:r-1
    R=u-tau*(2*((A)')*A*u-2*A'*b);
    u=P(R);
    U(:,itt+1)=u;
end

end

function y = ProjPositif( x )
y=x.*(x>0);
end

function y = ProjCyl( x )

y=[x(1:2).*(norm(x(1:2),2)<1) + x(1:2)/norm(x(1:2),2).*(norm(x(1:2),2) >= 1);x(3:end)];

end

function S = gradContraint( x,N,tau,A )
```

```
S=zeros(length(x),N);
x=x(:);
S(:,1)=x;

for itt=2:N
    X=S(:,itt-1);
    S(:,itt)=X-2/tau*gradJ2(X,A);
end

end

function Y = gradJ2(X,A)

x1=X(1);
x2=X(2);
Y=[2*x1-14+A.*((x1+x2-2)>=0)+A.*((x1+2*x2-3)>=0);2*x2-6+A.*((x1+x2-2)>=0)+2*A.*((x1+2*x2-3)>=

end
```

Application au traitement d'image.

Fichier lanceur (ou script).

```
%% Essais.
%% Chargement d'image.

load lena
figure
imagesc(Im)
axis square
colormap gray

%% Restauration, norme L2.
clc, clear all, close all

load lena
Im=lena;

Data=Im+30*randn(size(Im));
imagesc(Data)
axis square
colormap gray
axis off
%imwrite(Data,'lenna_bruit_l2_orig.png','png');

figure
D=[0 -1 0;
    -1 4 -1;
    0 -1 0];
I=descente(Data,1,D,500,1/80,1.2,Data);%function Im=descente(ImInit, T, D, N, param,lambda,da
imagesc(I)
axis square
colormap gray
axis off

%imwrite(I,'lenna_bruit_l2_orig_la1.png','png');

err=10*log10(Norm_2(Im)/Norm_2(Im-Data))
```

```
H=lin_IM(I,Im);
errf=10*log10(Norm_2(Im)/Norm_2(Im-H))

figure
I2=descente(Data,1,D,500,1/80,0.1,Data);%function Im=descente(ImInit, T, D, N, param,lambda,d
imagesc(I2)
axis square
colormap gray
axis off

%imwrite(I2,'lenna_bruit_l2_orig_la01.png','png');

%% Deconvolution norme 2.
clc, clear all, close all

k=15;sigma=14;
[X,Y]=meshgrid(-k:k);
Gauss=exp(-1/(2*sigma)*(X.^2+Y.^2));
surf(X,Y,Gauss,'EdgeColor','black')

load lena

Data=filter2(Gauss,Im);
Data=Data+100*randn(size(Im));

figure
imagesc(Data)
axis square
axis off
colormap gray
figure
D=[0 -1 0;
    -1 4 -1;
    0 -1 0];
I=descente(Im,Gauss,D,100,1/5000,2,Data);%I=descente(zeros(size(Im)),Gauss,D,100,1/500,1,Data
imagesc(I)
axis square
axis off
colormap gray

err=Norm_2(Im-Data)/Norm_2(Im)
H=lin_IM(I,Im);
erf=Norm_2(Im-H)/Norm_2(Im)

%% Inpainting.
clc, clear all, close all

load lena

d=75;
[u,v]=size(Im);
NM=u*v;
P=randperm(NM);
Im(P(1:(d*NM/100)))=0;

Mask=ones(size(Im));
Mask(P(1:(d*NM/100)))=0;
```

```
figure
imagesc(Im)
axis square
axis off
colormap gray
figure
D=[0 -1 0;
    -1 4 -1;
    0 -1 0];
I=descente2(Im,Mask,D,10000,1/200,1,Im);
I=descente2(I,Mask,D,10000,1/500,1,Im);
I=descente2(I,Mask,D,10000,1/1000,1,Im);
imagesc(I)
axis square
colormap gray
axis off


%%

T=-5:0.01:5;
F=sqrt(T.^2+0.2);
plot(T,F)
%title('Visualisation de la fonction L1-L2.')

%% L2-L1. Deconvolution.
clc, clear all, close all

k=15;sigma=14;
[X,Y]=meshgrid(-k:k);
Gauss=exp(-1/(2*sigma)*(X.^2+Y.^2));
surf(X,Y,Gauss,'EdgeColor','black')

load lena

Data=filter2(Gauss,Im);
Data=Data+100*randn(size(Im));

figure
imagesc(Data)
axis square
axis off
colormap gray
figure
D=[0 -1 0;
    -1 4 -1;
    0 -1 0];
I=descente3(zeros(size(Im)),Gauss,D,100,1/5000,200,Data,1);
imagesc(I)
axis square
axis off
colormap gray

err=Norm_2(Im-Data)/Norm_2(Im)
H=lin_IM(I,Im);
erf=Norm_2(Im-H)/Norm_2(Im)
```

```
%% L2-L1. Restauration.
clc, close all

load lena

Data=Im+30*randn(size(Im));

figure
imagesc(Data)
axis square
axis off
colormap gray
figure
D=[0 -1 0;
    -1 4 -1;
    0 -1 0];

%I=descente3(Data,1,D,50,1/10000,500000,Data,11);% Im=descente3(ImInit, T, D, N, param,lambda
%I=descente3(I,1,D,100,1/100,1500,Data,11);
I=descente3(Data,1,D,50,1/500,50700,Data,1);
imagesc(I)
axis square
axis off
colormap gray


err=10*log10(Norm_2(Im)/Norm_2(Im-Data))
H=lin_IM(I,Im);
errf=10*log10(Norm_2(Im)/Norm_2(Im-H))


%% L2-L1. impainting !!!Elapsed time is 399.185246 seconds.
clc, clear all, close all

load lena

d=75;
[u,v]=size(Im);
NM=u*v;
P=randperm(NM);
Im(P(1:(d*NM/100)))=0;

Mask=ones(size(Im));
Mask(P(1:(d*NM/100)))=0;

figure
imagesc(Im)
axis square
axis off
colormap gray
figure
D=[0 -1 0;
    -1 4 -1;
    0 -1 0];

tic
I=descente4(Im,Mask,D,300,1/10,2000,Im,10);
I=descente4(I,Mask,D,300,1/20,2000,Im,10);
```

```
I=descente4(I,Mask,D,100,1/30,2000,Im,10);
I=descente4(I,Mask,D,600,1/50,2000,Im,10);
toc

imagesc(I)
axis square
axis off
colormap gray

%% Algorithme de Chambolle.
clc, clear all, close all

load marisa
Im=marisa;

sigma=30;
Data=Im+sigma*randn(size(Im));

figure
imagesc(Data)
axis square
colormap gray

[I2,f]=debruitage_chambolle(Data,30,sigma,60);

figure
plot(f)

figure
imagesc(I2)
axis square
colormap gray

figure
imagesc(abs(I2-Data))
axis square
colormap gray

Norm_2(I2-Im)
%% Algorithme de Chambolle, simple.
%clc, clear all, close all

load lena

sigma=30;
%Data=Im+sigma*randn(size(Im));

figure
imagesc(Data)
axis square
colormap gray

tic
I=chambolle_simple(4000,30,Data);
toc

figure
imagesc(I)
```

```
axis square
colormap gray

figure
imagesc(abs(I-Data))
axis square
colormap gray

err=Norm_2(Im-Data)
errf=Norm_2(Im-I)

%% Algorithme de Chambolle ameliore.
clc, clear all, close all

load lena
Im=lena;

sigma=30;
Data=Im+sigma*randn(size(Im));

figure
imagesc(Data)
axis square
colormap gray
%%
sigma=40

tic
[I2,f]=debruitage_chambolle2(Data,47,sigma,10);
toc

figure
imagesc(Im)
axis square
colormap gray

figure
plot(f)

figure
imagesc(I2)
axis square
colormap gray
axis off

figure
imagesc(abs(I2-Data))
axis square
colormap gray

err=10*log10(Norm_2(Im)/Norm_2(Im-Data))
H=lin_IM(I2,Im);
errf=10*log10(Norm_2(Im)/Norm_2(Im-H))
%%
clc
A=[1,2,5;3,4,9;4,3,1]
[g1,g2]=gradient_chambolle(A)
U=divergence_chambolle(g1,g2)
```

```
%%
close all
T=[-100:0.1:0];
plot(T,-(T+1)./(T-1));
%% Forward-Backward.
 clear all, close all

load lena
Im=lena;

sigma=30;
Data=Im+sigma*randn(size(Im));

figure
imagesc(Data)
axis square
colormap gray

tic
I2=debruitage_chambolleFB(Data,20,sigma,10);
toc

figure
imagesc(Im)
axis square
colormap gray

figure
imagesc(I2)
axis square
colormap gray

figure
imagesc(abs(I2-Data))
axis square
colormap gray
Norm_2(I2-Im)

err=Norm_2(Im-Data)
errf=Norm_2(Im-I2)

%% Forward-Backward deconv.
 clear all, close all

load lena
Im=lena;
k=10;sigma=4;
[X,Y]=meshgrid(-k:k);
Gauss=exp(-1/(2*sigma)*(X.^2+Y.^2));
Data=filter2(Gauss,Im);
Data=Data+100*randn(size(Im));

figure
imagesc(Data)
axis square
colormap gray

tic
```

```
I2=deconv_chambolleFB( Data , 30, 8 ,10,0.002,Gauss);%deconv_chambolleFB( g , N, iterations_gr
toc

figure
imagesc(Im)
axis square
colormap gray
axis off

figure
imagesc(I2)
axis square
colormap gray
axis off

err=Norm_2(Im-Data)/Norm_2(Im)
errf=Norm_2(Im-I2)/Norm_2(Im)
%% Forward-Backward, impainting.
 clear all, close all

load lena
Im=lena;

h=length(Im(:));
Mask=zeros(size(Im));
R=randperm(h);
Mask(R(1:floor(h*0.50)))=1;

sigma=30;
Data=Im.*Mask;

figure
imagesc(Data)
axis square
colormap gray

tic
I2=inpainting_chambolleFB(Data,30,sigma,20,Mask);
toc

figure
imagesc(Im)
axis square
colormap gray

figure
imagesc(I2)
axis square
colormap gray

err=Norm_2(Im-Data)
errf=Norm_2(Im-I2)

%% Forward-Backward et beck teboulle
 clear all, close all

load lena
Im=lena;
```

```
sigma=30;
Data=Im+sigma*randn(size(Im));

figure
imagesc(Data)
axis square
colormap gray

tic
I2=debruitage_chambolle_FB_BT(Data,10,sigma,20);
toc

figure
imagesc(Im)
axis square
colormap gray

figure
imagesc(I2)
axis square
colormap gray

figure
imagesc(abs(I2-Data))
axis square
colormap gray

err=Norm_2(Im-Data)
errf=Norm_2(Im-I2)

%% Forward-Backward et Beck Teboulle applique a l'inpainting.
 clear all, close all

load lena
Im=lena;

h=length(Im(:));
Mask=zeros(size(Im));
R=randperm(h);
Mask(R(1:floor(h*0.50)))=1;

sigma=22;
Data=Im.*Mask;

figure
imagesc(Data)
axis square
colormap gray

tic
I2=inpainting_chambolle_FB_BT(Data,12,sigma,20,Mask);
toc

figure
imagesc(Im)
axis square
colormap gray
```

```
figure
imagesc(I2)
axis square
colormap gray

figure
imagesc(abs(I2-Data))
axis square
colormap gray

err=Norm_2(Im-Data)/Norm_2(Im)
errf=Norm_2(Im-I2)/Norm_2(Im)
%%
figure
imagesc(abs(I2-Data))
axis square
colormap gray

%%
k=10;
[X,Y]=meshgrid(-k:k);
P=3*X-Y;
surf(X,Y,P,'EdgeColor','black')

%% TVL1 bruit gaussien
 clear all, close all

load lena
Im=lena;

sigma=30;
Data=Im+sigma*randn(size(Im));

figure
imagesc(Data)
axis square
colormap gray
axis off

tic
I2=TVL1(Data,0.5,1,0.1,0.6,1000);%function x_1 = TVL1( Im,  tau , theta, sigma ,lambda , N )
toc
%
% figure
% imagesc(Im)
% axis square
% colormap gray

figure
imagesc(I2)
axis square
colormap gray
axis off

figure
imagesc(abs(I2-Data))
axis square
```

```
colormap gray
axis off
%% TVL1 bruit poivre et sel
clear all, close all

load lena
Im=lena;

sigma=100;
percent=25;
p=percent/100;
Data=Im;
h=length(Im(:));
R=randperm(h);
Data(R(1:floor(h*p)))=Data(R(1:floor(h*p)))+sigma*randn(1,floor(h*p));
IM= Data>255;
Data(IM)=255;
IM=find(Data<0);
Data(IM)=0;

figure
imagesc(Data)
axis square
colormap gray
axis off

tic
I2=TVL1(Data,5,1,0.01,0.6,200);%function x_1 = TVL1( Im,  tau , theta, sigma ,lambda , N )
toc%I2=TVL1(Data,5,1,0.01,0.18,200);

figure
imagesc(Im)
axis square
colormap gray

figure
imagesc(I2)
axis square
colormap gray
axis off

figure
imagesc(abs(I2-Data))
axis square
colormap gray
axis off

err=10*log10(Norm_2(Im)/Norm_2(Im-Data))
H=lin_IM(I2,Im);
errf=10*log10(Norm_2(Im)/Norm_2(Im-H))

%% homo
load homo
a=140;
I1=Im.*(Im<a)+255*(Im>=a);
figure
imagesc(I1)
axis square
```

```
colormap gray
axis off

M=(Im<a);
figure
imagesc(M)
axis square
colormap gray
axis off
M=double(M);
Im=I1.*M;

save('mask.txt', 'M', '-ASCII')
save('donnee.txt', 'Im', '-ASCII')

%% homo2
tic
I4=TV_inpainting(I1,5,1,0.02,1000,6000,M);%function x_1 = TV_inpainting( Im,
tau , theta, sigma ,lambda , N ,mask)
toc

figure
imagesc(I4)
axis square
axis off
colormap gray

%% primal dual applique a l'inpainting.
 clear all, close all

load lena
Im=lena;

 h=length(Im(:));%masque aleatoire
 Mask=zeros(size(Im));%masque aleatoire
 R=randperm(h);%masque aleatoire
 Mask(R(1:floor(h*0.10)))=1;%masque aleatoire
% Mask=ones(size(Im));%masque carr
% Mask(240:270,240:270)=0;%masque carr

sigma=22;
Data=Im.*Mask;

figure
imagesc(Data)
axis square
axis off
colormap gray
title('0.03')

figure

tic
I2=TV_inpainting(Data,5,1,0.02,1000,15000,Mask);%function x_1 = TV_inpainting( Im,
tau , theta, sigma ,lambda , N ,mask)
toc

figure
```

```
imagesc(Im)
axis square
axis off
colormap gray

figure
imagesc(I2)
axis square
axis off
colormap gray

figure
imagesc(abs(I2-Data))
axis square
axis off
colormap gray

err=Norm_2(Im-Data)/Norm_2(Im)
errf=Norm_2(Im-I2)/Norm_2(Im)

%Elapsed time is 468.875980 seconds. pour 90 (8 min)%
%% primal dual applique a l'inpainting.

load lena
Im=lena;
%
h=length(Im(:));%masque aleatoire
Mask=zeros(size(Im));%masque aleatoire
R=randperm(h);%masque aleatoire
Mask(R(1:floor(h*0.05)))=1;%masque aleatoire

Data=Im.*Mask;

tic
I4=TV_inpainting(Data,5,1,0.02,1000,6000,Mask);%function x_1 = TV_inpainting( Im,
tau , theta, sigma ,lambda , N ,mask)
toc

figure
imagesc(I4)
axis square
axis off
colormap gray
%%
tau=5 , theta=1, sigma =0.02,lambda=1000 , N=6000

%%
%[a,b]=gradient_chambolle(Im);
%I4=divergence_chambolle(a,b);
%
figure
imagesc(b)
axis square
axis off
colormap gray
%% date de l'agreg
temps=1341821983;
```

```
c=32768;
b=612;


for N=temps:temps+(100*3600)

    A=factor(N);
    P=unique(A);
    a=histc(A,P);

    a1=prod(a.^P);
    b1=sum(a.*P);

    if ((a1==c) && (b1==b))
        TT=N
    end
end


%%
A=factor(9*4)
U=unique(A)
n=histc(A,U)

%%

A=factor(TT)
P=unique(A)
a=histc(A,P)

a1=prod(a.^P)
b1=sum(a.*P)

((a1==c) & (b1==b))
%    TT=N
% end

%%
clc

T=[1.0   2.0   6.0   1.0
6.0   0.0   2.0   1.0
6.0   0.0   3.0   1.0
4.0   4.0   5.0   3.0
2.0   1.0   7.0   1.0   ]
[a,b]=gradient_chambolle(T)
S=[5.0   -2.0   -4.0   0.0
0.0   0.0   1.0   0.0
-2.0   4.0   2.0   2.0
-2.0   -3.0   2.0   -2.0
0.0   0.0   0.0   0.0          ]
a==S
W=[1.0   4.0   -5.0   0.0
-6.0   2.0   -1.0   0.0
-6.0   3.0   -2.0   0.0
0.0   1.0   -2.0   0.0
-1.0   6.0   -6.0   0.0   ]
W==b
```

```
V=divergence_chambolle(a,b)
D=[6.0   1.0   -13.0   5.0
-11.0   10.0   2.0   1.0
-8.0   13.0   -4.0   4.0
0.0   -6.0   -3.0   -2.0
1.0   10.0   -14.0   8.0    ]
D==V

%% masque
load lena
Im=lena;
%
h=length(Im(:));%masque aleatoire
Mask=zeros(size(Im));%masque aleatoire
R=randperm(h);%masque aleatoire
Mask(R(1:floor(h*0.1)))=1;%masque aleatoire
Im=Im.*m;
save('mask.txt', 'm', '-ASCII')
save('donnee.txt', 'Im', '-ASCII')
figure
imagesc(Im)
axis 'square'
colormap gray
axis off
%% lecture du fichier C.
ide=fopen('sortie.txt');
A=fscanf(ide,'%g');
A=reshape(A,512,512);
figure
imagesc(A')
axis 'square'
colormap gray
axis off

%%
close all
load lena
figure
imagesc(Im)
axis square off
colormap gray

[u,v]=gradient_chambolle(Im);
A=sqrt(u.^2+v.^2);

%figure
%imagesc(A);
%axis square off
%colormap gray

a=A(:);
[B,I]=sort(a);
C=A;
p=0.99;
C(I(1:floor(p*length(C(:)))))=0;
m=ones(size(C));
m(I(1:floor(p*length(C(:)))))=0;
```

```
figure
imagesc(C);
axis square off
colormap gray

figure
imagesc(m);
axis square off
colormap gray
save('mask.txt', 'Mask', '-ASCII')

figure
plot(B(:))

%% homo2
load bergman
A=imread('homo2.png');
Im=C(1)*A(:,:,1)+C(3)*A(:,:,3)+C(2)*A(:,:,2);
Im=double(Im);

tic
I2=TVL1(Im,2,1,0.2,0.7,800);%function x_1 = TVL1( Im,  tau , theta, sigma ,lambda , N )
toc%I2=TVL1(Data,5,1,0.01,0.18,200);

figure
imagesc(Im)
axis off
colormap gray

figure
imagesc(I2/160*250);
colormap gray
axis off

%% test masques
clc, close all, clear all;
load morgan
Im=morgan;

fig(Im)

[u,v]=gradient_chambolle(Im);

g=sqrt(u.^2+v.^2);
fig(g);
a=sort(g(:));


[x,y]=meshgrid([-6:6],[-6:6]);
F=exp(-1/16*(x.^2+y.^2));
F=F/sum(F(:));

g=filter2(F,g);
fig(g);

M=g./max(a);
```

```
R=rand(size(M));
O=double(R<(M*0.4));

fig(O);


p=100*(1-sum(sum(O))/length(a))

Data=Im.*O;


save('mask.txt', 'O', '-ASCII')
save('donnee.txt', 'Data', '-ASCII')

fig(Data);
%% lecture du fichier C.

load reussite2
ide=fopen('sortie.txt');
A=fscanf(ide,'%g');
A=reshape(A,512,512);
fig(A')

fig(Data)
E=A'-Im;
err=norm(E(:),2)
fig(E)
fig(Im)
%% remarquable
close all
hold on
load lena
[u,v]=gradient_chambolle(Im);
A=u.^2+v.^2;
B=abs(filter2([-1,0;2,-1],Im));
B=B(:);
A=A(:);
c=sort(A);
d=sort(B);
plot(c);
plot(d/max(B)*max(A),'r');
figure
hist(A,100);
figure
hist(B/max(B)*max(A),100);

% load bergman
% [u,v]=gradient_chambolle(bergman);
% A=u.^2+v.^2;
% A=A(:);
% d=sort(A);
% plot(d,'r');

% load morgan
% [u,v]=gradient_chambolle(morgan);
% A=u.^2+v.^2;
% A=A(:);
% e=sort(A);
```

```
% plot(e,'g');
%
% load marisa
% [u,v]=gradient_chambolle(marisa);
% A=u.^2+v.^2;
% A=A(:);
% f=sort(A);
% plot(f,'c');
%
% load segolene
% Im=Im(1:512,1:512);
% [u,v]=gradient_chambolle(Im);
% A=u.^2+v.^2;
% A=A(:);
% f=sort(A);
% plot(f,'m');
%
% Im=20*randn(512,512)+120;
% [u,v]=gradient_chambolle(Im);
% A=u.^2+v.^2;
% A=A(:);
% f=sort(A);
% plot(f,'y');
%%

load marisa
[u,v]=gradient_chambolle(marisa);
A=u.^2+v.^2;
A=A(:);
f=sort(A);
plot(1./f,'c');
Inv=1./f;
hold on
xx=1:length(Inv);
plot(f)

x = [250000:250025]
y = f(x);%x.^3
plot(xx,csapi(x,y,xx),'k-',x,y,'ro')
title('Interpolant to Three Points')
%%
clear all
close all
load marisa
imagesc(marisa);
colormap gray

f=fft2(marisa);
figure
imagesc(log(1+abs(f)));
colormap gray

N=2048*2;
F=zeros(N,N);
F(N/2-256:N/2+255,N/2-256:N/2+255)=fftshift(f);


figure
```

```
imagesc(abs(ifft2(fftshift(F))));
colormap gray
```

Optimisation lisse.

```
function Im=descente(ImInit, T, D, N, param,lambda,data)
%function Im=descente(ImInit, T, D, N, param,lambda,data)
Im=ImInit;
Is=zeros(size(Im));
%Im=Im.*(Im>=0).*(Im<=255)+255*ones(size(Im)).*(Im>255)+zeros(size(Im)).*(Im<0);

while(Norm_2(Is-Im)>0.1)

% for itt=1:N
Is=Im;
      Im=Im-param*GradientDT(Im,D,T,lambda,data);
%      %Im=Im.*(Im>=0).*(Im<=255)+255*ones(size(Im)).*(Im>255)+zeros(size(Im)).*(Im<0);

%      imagesc(Im)
%      axis square
%      colormap gray
%      pause(0.01)


end

end

function Im=descente2(ImInit, T, D, N, param,lambda,data)

Im=ImInit;

for itt=1:N
    Im=Im-param*GradientDT2(Im,D,T,lambda,data);
end

end

function Im=descente3(ImInit, T, D, N, param,lambda,data,epsilon)
% Im=descente3(ImInit, T, D, N, param,lambda,data,epsilon)
Im=ImInit;

for itt=1:N
    Im=Im-param*GradientDT3(Im,D,T,lambda,data,epsilon);

    imagesc(Im)
    axis square
    colormap gray
    pause(0.01)
end

end

function Im=descente4(ImInit, T, D, N, param,lambda,data,epsilon)

Im=ImInit;

for itt=1:N
    Im=Im-param*GradientDT4(Im,D,T,lambda,data,epsilon);
```

```
end

end
```

Optimisation non lisse.

```
function Im=chambolle_simple( N,lambda,g)
%function Im=chambolle( N,lambda,g)
tau=1/5;
Im1=zeros(size(g));
Im2=zeros(size(g));

%Itérations de l'agorithme de Chambolle.
for itt=1:N
    [G1,G2]=gradient_chambolle(divergence_chambolle(Im1,Im2)-g/lambda);
    G=sqrt(G1.^2+G2.^2);
    Im1=(Im1+tau*G1)./(1+tau*G);
    Im2=(Im2+tau*G2)./(1+tau*G);
end

Im=g-lambda*divergence_chambolle(Im1,Im2);

end

function [v,F] = debruitage_chambolle( g , N, sigma,iterations_gradient)
% function Im = debruitage_chambolle( g , N, sigma,iterations_gradient)


Npix=floor(sqrt(length(g(:))));
tau=1/5;
lambda=sigma;
v=zeros(size(g));
F=zeros(1,N);

for itt=1:N
    v=chambolle_simple2(v,iterations_gradient,lambda,g,tau);
    f=sqrt(sum(sum((v-g).^2)));
    F(itt)=f;
    lambda=(Npix*sigma/f)*lambda;
end

end

function [v,F] = debruitage_chambolle2( g , N, sigma,iterations_gradient)
% function Im = debruitage_chambolle2( g , N, sigma,iterations_gradient)


Npix=floor(sqrt(length(g(:))));
tau=1/5;
lambda=sigma;
v=zeros(size(g));
F=zeros(1,N);
Im1=zeros(size(g));
Im2=zeros(size(g));

for itt=1:N

    for itt2=1:iterations_gradient
        [G1,G2]=gradient_chambolle(divergence_chambolle(Im1,Im2)-g/lambda);
```

```
        G=sqrt(G1.^2+G2.^2);
        Im1=(Im1+tau*G1)./(1+tau*G);
        Im2=(Im2+tau*G2)./(1+tau*G);
    end

    v=g-lambda*divergence_chambolle(Im1,Im2);
    f=sqrt(sum(sum((v-g).^2)));
    F(itt)=f;
    lambda=(Npix*sigma/f)*lambda;
end

end

function [v] = debruitage_chambolleFB( g , N, sigma,iterations_gradient)
% function Im = debruitage_chambolle( g , N, sigma,iterations_gradient)


lambda=sigma/4;
v=zeros(size(g));
gamma=1.8;

for itt=1:N
    v=v-gamma*(v-g)/lambda;
    v=chambolle_simple(iterations_gradient,lambda,v);
end

end

function [v] = debruitage_chambolleFB( g , N, sigma,iterations_gradient)
% function Im = debruitage_chambolle( g , N, sigma,iterations_gradient)


lambda=sigma/4;
v=zeros(size(g));
gamma=1.8;

for itt=1:N
    v=v-gamma*(v-g)/lambda;
    v=chambolle_simple(iterations_gradient,lambda,v);
end

end

function x_1 = TVL1( Im,  tau , theta, sigma ,lambda , N )
% function x_1 = TVL1( Im,  tau , theta, sigma ,lambda , N )
x=Im;
x_bar=x;
y1=zeros(size(Im));
y2=zeros(size(Im));

for i=1:N
    [a,b]=gradient_chambolle(x_bar);
    t1=y1-sigma*a;
    t2=y2-sigma*b;
    norm=(Norm_I(t1,t2));
    y1=t1/max(1,norm);
    y2=t2/max(1,norm);
    t=x-tau * divergence_chambolle(y1,y2);
```

```
    x_1= (t-tau*lambda).*((t-Im)>(tau*lambda)) + (t+tau*lambda).*((t-Im)<(-tau*lambda)) + Im.


    x_bar = x_1 + theta*(x_1-x);
%
%     imagesc(x_1)
%     axis square
%     colormap gray
%     pause(0.1)

    x=x_1;
end

end

function t = TV_inpainting( Im,  tau , theta, sigma ,lambda , N ,mask)
%#codegen
% function x_1 = TV_inpainting( Im,  tau , theta, sigma ,lambda , N ,mask)
x=Im;
x_bar=x;
y1=zeros(size(Im));
y2=zeros(size(Im));

for i=1:N
    [a,b]=gradient_chambolle(x_bar);
    t1=y1-sigma*a;
    t2=y2-sigma*b;
    norm=(Norm_I(t1,t2));
    %M=max(1,norm);%
    y1=t1/max(1,norm);
    y2=t2/max(1,norm);
    t=x-tau * divergence_chambolle(y1,y2);

    x_1=(ones(size(mask))-mask).*t + mask.*(t+tau*lambda*Im)/(1+tau*lambda);

    x_bar = x_1 + theta*(x_1-x);

%     imagesc(x_1)
%     axis square
%     axis off
%     colormap gray
%     pause(0.01)

    x=x_1;
end

end

function [v] = inpainting_chambolleFB( g , N, sigma,iterations_gradient,mask)
% function [v] = inpainting_chambolleFB( g , N, sigma,iterations_gradient,mask)

lambda=sigma/4;
v=zeros(size(g));

for itt=1:N
    v=g.*mask+(1-mask).*v;%v+mask.*(g-v)
```

```
    v=chambolle_simple(iterations_gradient,lambda,v);
end

end

function x = inpainting_chambolle_FB_BT( g , N, sigma,iterations_gradient,mask)
% function x = inpainting_chambolle_FB_BT( g , N, sigma,iterations_gradient,mask)
lambda=sigma/4;
v=zeros(size(g));
beta=0.6;

x=v;
z=x;
t=1;

for itt=1:N
    y=g.*mask*1/(lambda*beta)+(ones(size(mask))-mask*1/(lambda*beta)).*z;
    x_1=chambolle_simple(iterations_gradient,lambda/beta,y);
    t_1=(1+sqrt(4*t^2+1))/2;
    lambda_n=1+(t-1)/t_1;
    t=t_1;
    z=x+lambda_n*(x_1-x);
    x=x_1;
end

end

function [v] = deconv_chambolleFB( g , N, iterations_gradient,lambda,gamma,T)
% function [v] = deconv_chambolleFB( g , N, iterations_gradient,lambda,gamma,T)

v=g;
Tt=rot90(T,2);

for itt=1:N
     v=v+gamma*filter2(Tt,2*g-filter2(T,v));
     v=chambolle_simple(iterations_gradient,lambda,v);
end

end
```

Implémentation de l'impainting par minimisation de la variation totale en JAVA.

Fichier principal.

```
package ImageTV;

public class principal {

	/**
	 * -i 200 -t -o "C:\\toto.png" -m
	 * "C:\\Documents and Settings\\fabien\\Mes documents\\Stage\\mask.png"
	 * "C:\\Documents and Settings\\fabien\\Mes documents\\Stage\\Lenna.png"
	 */
	public static void main(String[] args) {

		@SuppressWarnings("unused")
		OptionParser opt = new OptionParser(args);

		MyImage mi = new MyImage(Options.inputFilename);

		/* Capture du masque.*/
		MyImage mask = new MyImage(Options.inputFileMaskname);
		mask.setMask();

		MyImage masked = MyImage.mutiplyImage(mi, mask);

		long temps_init = System.currentTimeMillis();

		if(Options.verbose){
			System.out.println("Le programme va tourner " + Options.itterations
					+ "  i t t rations .");
		}

		MyImage toto = MyImage.IterationTV(masked,5,1,0.02,1000,
				Options.itterations,mask);

		if(Options.verbose){
			System.out.println("Le programme a       e x cut en " +
		((System.currentTimeMillis()-temps_init)/1000) + " s.");
		}

		toto.write(Options.outputFilename);
		System.exit(0);
	}
}
```

Fichier classe Gradient.

```
package ImageTV;

public class Gradient {

	double[] gradX;
	double[] gradY;
	int height;
	int width;
	int N;
```

```
Gradient(MyImage monImage){
        N=monImage.length;
        height=monImage.height;
        width=monImage.width;
        gradX= new double[monImage.length];
        gradY= new double[monImage.length];

        for(int col=0; col<monImage.width; col++){
                for(int lin=0; lin<monImage.height; lin++){
                        if(lin==(monImage.height-1)){
                                gradX[lin+col*monImage.height]=0;
                        }
                        else{
                                gradX[lin+col*monImage.height]=
                                        monImage.BW[lin+(col)*monImage.height
                                                -monImage.BW[lin+col*
                        }
                        if((col==(monImage.width-1))){
                                gradY[lin+col*monImage.height]= 0;
                        }
                        else{
                                gradY[lin+col*monImage.height]=
                                        monImage.BW[lin+(col+1)*monImage.heig
                                                -monImage.BW[lin+col*
                        }
                }
        }
}


void resetGradient(MyImage monImage){
        N=monImage.length;
        height=monImage.height;
        width=monImage.height;
        gradX= new double[monImage.length];
        gradY= new double[monImage.length];
        for(int col=0; col<monImage.width; col++){
                for(int lin=0; lin<monImage.height; lin++){
                        if(lin==(monImage.height-1)){
                                gradX[lin+col*monImage.height]=0;
                        }
                        else{
                                gradX[lin+col*monImage.height]=
                                        monImage.BW[lin+(col)*monImage.height
                                                -monImage.BW[lin+col*
                        }
                        if((col==(monImage.width-1))){
                                gradY[lin+col*monImage.height]= 0;
                        }
                        else{
                                gradY[lin+col*monImage.height]=
                                        monImage.BW[lin+(col+1)*monImage.heig
                                                -monImage.BW[lin+col*
                        }
                }
        }
}
```

```
static double[] divergence(Gradient grad){
        double[] result = new double[grad.N];
        double x_1;
        double x_2;
        for(int lin=0; lin<grad.height; lin++){
                for(int col=0; col<grad.width; col++){
                        if(lin==0){
                                x_1=grad.gradX[lin+grad.height*col];
                        }
                        else if (lin==(grad.height-1)) {
                                x_1=-grad.gradX[lin+grad.height*col-1];
                        }
                        else{
                                x_1=grad.gradX[lin+grad.height*col]
                                                -grad.gradX[lin+grad.height*col-1];
                        }

                        if (col==0) {
                                x_2=grad.gradY[lin+grad.height*col];
                        }
                        else if (col==(grad.width-1)) {
                                x_2=-grad.gradY[lin+grad.height*(col-1)];
                        }
                        else{
                                x_2=grad.gradY[lin+grad.height*col]
                                                -grad.gradY[lin+grad.height*(col-1)];
                        }
                        result[lin+grad.height*col]= x_1+x_2;
                }
        }
        return result;
}


void multiplyGradient(double lambda){
        for(int i=0; i<N; i++){
                gradX[i]= lambda*gradX[i];
                gradY[i]= lambda*gradY[i];
        }
}


double getNormI(){
        double temp;
        double m_rel=0;
        for(int i=0; i<N; i++){
                temp=Math.sqrt(gradX[i]*gradX[i]
                                + gradY[i]*gradY[i]);
                if(temp>m_rel){
                        m_rel=temp;
                }
        }
        return m_rel;
}


static double getNormI(double[] a, double[] b){
```

```
                double temp;
                double m_rel=0;
                for(int i=0; i<a.length; i++){
                        temp=Math.sqrt(a[i]*a[i] + b[i]*b[i]);
                        if(temp>m_rel){
                                m_rel=temp;
                        }
                }
                return m_rel;
        }
}
```

Fichier classe OptionParser.

```
package ImageTV;

import java.util.Scanner;

public class OptionParser {
        boolean isharg;
        boolean isVarg;

        OptionParser(String[] args){
                Options.debug=false;
                Options.verbose=false;
                isharg =false;
                isVarg =false;
                for(int optind =0; optind<args.length; optind++){
                        Scanner s = new Scanner(args[optind]);
                        String argument = args[optind];
                        if(optind==args.length-1){
                                Options.inputFilename= new String(args[optind]);
                        }
                        if(argument.charAt(0) == '-'){
                                switch (argument.charAt(1)) {
                                case 'h':
                                        isharg=true;
                                        break;

                                case 'd':
                                        Options.debug=true;
                                        break;

                                case 't':
                                        Options.verbose=true;
                                        break;


                                case 'V':
                                        isVarg=true;
                                        break;

                                case 'o':
                                        if(args[optind+1].charAt(0) == '-'){
                                                System.out.println("Erreur : Pas de fichier d
                                                System.exit(1);
                                        }
                                        else{
```

```
                                Options.outputFilename=new String(args[++opti
                        }
                        break;


                case 'm':
                        if(args[optind+1].charAt(0) == '-'){
                                System.out.println("Erreur : Pas de masque !"
                                System.exit(1);
                        }
                        else{
                                Options.inputFileMaskname=new String(args[++o
                        }
                        break;

                case 'i':
                        if(args[optind+1].charAt(0) == '-'){
                                Options.itterations=6000;
                        }
                        else{
                                Integer t = new Integer(args[++optind]);
                                Options.itterations=t.intValue();
                        }
                        break;
                case '-':
                        if(s.next().contentEquals("--help")){
                                isharg=true;
                        }
                        if(s.next().contentEquals("--mask")){
                                Options.inputFileMaskname=new String(args[++o
                        }

                        if(s.next().contentEquals("--itt")){
                                Integer t = new Integer(args[++optind]);
                                Options.itterations=t.intValue();
                        }
                        if(s.next().contentEquals("--time")){
                                Options.verbose=true;
                        }
                        if(s.next().contentEquals("--version")){
                                isVarg=true;
                        }
                        if(s.next().contentEquals("--ouput")){
                                Options.outputFilename=new String(args[optind
                        }
                        break;

                default:
                        break;
                }
        }
}
if(isVarg){
        version();
}

if(isharg){
        help();
```

```
                }
        }

        static void help()
        {
                System.out.println("Le programme TvInpainting permet de retrouver une image n
                System.out.println( "Usage : %s [OPTION] FILE...." );
                System.out.println( "-m, --mask=FILE\tPrend le masque sur le fichier FILE.");
                System.out.println( "-o, --output=FILE\ t crit  le  r sultat  sur FILE.");
                System.out.println( "-t, --time\t\tAffichage du temps.");
                System.out.println( "-V, --version\t\tdisplay version and exit. ");
                System.out.println( "-h, --help\t\tAfficher l'aide.");
                System.exit(0);
        }

        static void version()
        {
                System.out.println( "Version 1.");
                System.exit(0);
        }

}
```

Fichier MyImage.

```
package ImageTV;

import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.imageio.stream.FileImageInputStream;
import javax.imageio.stream.FileImageOutputStream;


public class MyImage {

        //Variational gradient;

        BufferedImage monImage;


        int width;
        int height;
        int length;

        double[] BW ;

        void setTestMyImage(){
                height=5;
                width=4;
                length=height*width;
                //gradient = new Variational(this);

                BW[0]=1;
```

```
                BW[1]=6;
                BW[2]=6;
                BW[3]=4;
                BW[4]=2;
                BW[5]=2;
                BW[6]=0;
                BW[7]=0;
                BW[8]=4;
                BW[9]=1;
                BW[10]=6;
                BW[11]=2;
                BW[12]=3;
                BW[13]=5;
                BW[14]=7;
                BW[15]=1;
                BW[16]=1;
                BW[17]=1;
                BW[18]=3;
                BW[19]=1;
        }


        MyImage(String args){
                File fichier = new File(args);
                try {
                        @SuppressWarnings("unused")
                        FileImageInputStream fichier_image = new FileImageInputStream(fichier
                } catch (FileNotFoundException e) {
                        System.out.println("Erreur : je n'ai pas trouve le fichier "
+ args);
                        e.printStackTrace();
                } catch (IOException e) {
                        System.out.println("Erreur : je n'ai pas trouve le fichier "
+ args);
                        e.printStackTrace();
                }
                monImage = null;
                try {
                        monImage = ImageIO.read(fichier);
                } catch (IOException e) {
                        System.out.println("Erreur : je n'ai pas trouve le fichier "
+ args);
                        e.printStackTrace();
                }

                width = monImage.getWidth();
                height = monImage.getHeight();
                int N=width*height;
                length=N;
                int[] pixels = new int[width*height];
                pixels = monImage.getRGB(0, 0, width, height, pixels, 0, width);

                BW = new double[N];
                Color c;
                for(int i=0; i<N; i++)
                {
                        c = new Color(pixels[i]);
                        BW[i]= 0.299 * c.getRed() + 0.587 * c.getGreen() + 0.114 * c.getBlue(
```

```
        }
        //@SuppressWarnings("unused")
        //Variational V= new Variational(this);
}


MyImage(MyImage im){
        width=im.width;
        height=im.height;
        length=width*height;
        BW=new double[length];
        for(int i=0; i<length; i++){
                BW[i]=im.BW[i];
        }
        monImage = im.monImage;
}

MyImage(int w, int h, MyImage im){
        width=w;
        height=h;
        length=w*h;
        BW= new double[length];
        for(int i=0; i<length; i++){
                BW[i]=im.BW[i];
        }
        monImage= im.monImage;
}

MyImage(int w, int h){
        width=w;
        height=h;
        length=w*h;
        BW= new double[length];
        for(int i=0; i<length; i++){
                BW[i]=0;
        }
        //gradient = new Variational(this);
}

void setMask(){
        for(int i=0; i<length; i++){
                if(BW[i]==0){
                        BW[i]=0;
                }
                else
                {
                        BW[i]=1;
                }
        }
}



int getLength(){
        return length;
}

double getPixel(int i){
```

```
        if(i<length & i>=0)
                return BW[i];
        else
                return(-1);
}

double getPixel(int x, int y){
        if(x<width & y<height & x>=0 & y>=0)
                return BW[x*height + y];
        else
                return(-1);
}

static MyImage additionImage(MyImage im1, MyImage im2){
        if((im1.height == im2.height)&&(im1.width == im2.width)){
                MyImage result = new MyImage(im1.height, im1.width, im1);
                for(int i=0; i<result.length; i++){
                        result.BW[i]= im1.BW[i]+im2.BW[i];
                }
                return result;
        }
        return null;
}


static MyImage additionToDoubleImage(MyImage im, double[] grad){
        if(im.length == grad.length){
                MyImage result = new MyImage(im.height, im.width, im);
                for(int i=0; i<result.length; i++){
                        result.BW[i]= im.BW[i]+grad[i];
                }
                return result;
        }
        return null;
}


void multiplyImage(double lambda){
        for(int i=0; i<length; i++){
                BW[i]= lambda*BW[i];
        }
}

double getNorm2(){
        double cumul=0;
        for(int i=0; i<length; i++){
                cumul+=BW[i];
        }
        return(Math.sqrt(cumul));
}

static MyImage mutiplyImage(MyImage im1, MyImage im2){
        if((im1.height == im2.height)&&(im1.width == im2.width)){
                MyImage result = new MyImage(im1.height, im1.width, im1);
                for(int i=0; i<result.length; i++){
                        result.BW[i]= im1.BW[i]*im2.BW[i];
                }
                return result;
```

```
        }
        return im2;
    }

    static MyImage invImage(MyImage im1){
        MyImage result = new MyImage(im1.height, im1.width, im1);
        for(int i=0; i<result.length; i++){
            result.BW[i]= 1-im1.BW[i];
        }
        return result;
    }


    static MyImage lambdaImage(MyImage im1, double lambda){
        MyImage result = new MyImage(im1.height, im1.width,im1);
        for(int i=0; i<result.length; i++){
            result.BW[i]= lambda*im1.BW[i];
        }
        return result;
    }

    void printMyImage(){
        System.out.println("La hauteur est '" + height + "'.");
        System.out.println("La largeur est '" + width + "'.");
            for(int l=0; l<height; l++){
        for(int c=0; c<width; c++){
                    System.out.print(BW[l+ c*height] + "
");
                }
                System.out.println(" ");
        }
    }

    public static MyImage test(MyImage Im){
        MyImage x= new MyImage(Im);
        Gradient g= new Gradient(x);
        x.BW= Gradient.divergence(g);
        for(int pix=0; pix<x.length; pix++){
            x.BW[pix]=g.gradY[pix];
        }
        return x;
    }

    void normalize(){
        double max_rel=0;
        double min_rel=255;
        for(int pix=0; pix<length; pix++){
            if(BW[pix]>max_rel){
                max_rel=BW[pix];
            }
            if(BW[pix]<min_rel){
                min_rel=BW[pix];
            }

        }

        double a=(255/(max_rel-min_rel));
        double b=255-a*max_rel;
```

```
        for(int pix=0; pix<length; pix++){
                BW[pix]=a*BW[pix]+b;
        }
}

public static MyImage IterationTV(MyImage Im, double tau , double theta,
                double sigma ,double lambda , int itterations ,MyImage mask){
        //function x_1 = TV_inpainting( Im,  tau , theta, sigma ,
        //lambda , N ,mask)
        MyImage x= new MyImage(Im);//x=Im;
        MyImage x_bar = new MyImage(x);//x_bar=x;

        MyImage y1= new MyImage(Im.width, Im.height);//y1=zeros(size(Im));
        MyImage y2= new MyImage(Im.width, Im.height);//y2=zeros(size(Im));
        MyImage t1= new MyImage(Im.width, Im.height, Im);
        MyImage t2= new MyImage(Im.width, Im.height, Im);
        MyImage t = new MyImage(Im);
        MyImage x_1 = new MyImage(Im);
        double normi;
        double normalizer;

        Gradient g= new Gradient(x_bar);

        for(int i=0; i<itterations; i++){//for i=1:N
                g.resetGradient(x_bar);//[a,b]=gradient_chambolle(x_bar);

                for(int pix=0; pix<x_1.length; pix++){
                        t1.BW[pix]= y1.BW[pix] - sigma * g.gradX[pix];//t1=y1-sigma*a
                        t2.BW[pix]= y2.BW[pix] - sigma * g.gradY[pix];//t2=y2-sigma*b
                }

                normi= Gradient.getNormI(t1.BW, t2.BW);
                if(1>normi){
                        normalizer=1;
                }
                else{
                        normalizer=normi;
                }

                for(int pix=0; pix<x_1.length; pix++){
                        y1.BW[pix]=t1.BW[pix]/normalizer;//y1=t1/max(1,norm);
                        y2.BW[pix]=t2.BW[pix]/normalizer;//y2=t2/max(1,norm);
                        g.gradX[pix]=y1.BW[pix];//********
                        g.gradY[pix]=y2.BW[pix] ;//********
                }

                double[] inter = Gradient.divergence(g);

                //t=x-tau * divergence_chambolle(y1,y2);
                for(int pix=0; pix<x_1.length; pix++){
                        t.BW[pix]=x.BW[pix]-tau*inter[pix];
                        //System.out.println(i+"Vinter=" +t.BW[pix]);
                }

                //x_1=(ones(size(mask))-mask).*t +
                //mask.*(t+tau*lambda*Im)/(1+tau*lambda);
                for(int pix=0; pix<x_1.length; pix++){
```

```
                                if(mask.BW[pix]==0){
                                        x_1.BW[pix]=t.BW[pix];
                                }
                                else{
                                        x_1.BW[pix]=(t.BW[pix]+ tau*lambda*Im.BW[pix])
                                                        /(1+tau*lambda);
                                }
                        }

                        //x_bar = x_1 + theta*(x_1-x);
                        for(int pix=0; pix<x_1.length; pix++){
                                x_bar.BW[pix]= x_1.BW[pix] + theta * (x_1.BW[pix] - x.BW[pix]
                        }

                        //x=x_1;
                        for(int pix=0; pix<x_1.length; pix++){
                                x.BW[pix]=x_1.BW[pix];
                        }
                }
                return x_1;
        }


        void write(String name){
                File fichier = new File(name);
                FileImageOutputStream fios = null;
                try {
                        fios = new FileImageOutputStream(fichier);
                } catch (FileNotFoundException e1) {
                        System.out.println("Erreur d'ouverture de fichier de sortie.");
                        e1.printStackTrace();
                } catch (IOException e1) {
                        System.out.println("Erreur d'ouverture de fichier de sortie.");
                        e1.printStackTrace();
                }

                int pix[]= new int[length];
                for(int i=0; i<length; i++){
                        pix[i] =  (((int)(BW[i]) << 16) + ((int)(BW[i]) << 8)
                                        + (int)(BW[i]));
                }

                monImage.setRGB(0, 0, width, height, pix, 0, width);

                try {
                        @SuppressWarnings("unused")
                        boolean r = ImageIO.write(monImage, "png", fios);
                } catch (IOException e) {
                        System.out.println("Erreur : je n'ai pas pu  rire  le fichier "
                + name);
                        e.printStackTrace();
                }
        }
}
```

Fichier Options.

```
package ImageTV;
```

```
public class Options {
        static boolean debug;
        static boolean verbose;
        static String inputFilename;
        static String inputFileMaskname;
        static String outputFilename;
        static int itterations;
}
```

Implémentation de l'impainting par minimisation de la variation totale en langage C.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


int height ;
int width ;
int N ;


void gradient(double * image_initiale, double ** gradient)
{
    for(int col=0; col<width; col++)
    {
        for(int lin=0; lin<height; lin++)
        {
            if(lin==(height-1))
            {
                gradient[0][lin+col*height]=0;
            }
            else
            {
                gradient[0][lin+col*height]=
                    image_initiale[lin+(col)*height+1]
                    -image_initiale[lin+col*height];
            }
            if((col==(width-1)))
            {
                gradient[1][lin+col*height]= 0;
            }
            else
            {
                gradient[1][lin+col*height]=
                    image_initiale[lin+(col+1)*height]
                    -image_initiale[lin+col*height];
            }
        }
    }
}


void divergence(double ** gradient, double * result)
{
    double x_1;
    double x_2;
    for(int lin=0; lin<height; lin++)
    {
        for(int col=0; col<width; col++)
        {
            if(lin==0)
            {
                x_1=gradient[0][lin+height*col];
            }
            else if (lin==(height-1))
            {
```

```
                x_1=-gradient[0][lin+height*col-1];
            }
            else
            {
                x_1=gradient[0][lin+height*col]
                    -gradient[0][lin+height*col-1];
            }

            if (col==0)
            {
                x_2=gradient[1][lin+height*col];
            }
            else if (col==(width-1))
            {
                x_2=-gradient[1][lin+height*(col-1)];
            }
            else
            {
                x_2=gradient[1][lin+height*col]
                    -gradient[1][lin+height*(col-1)];
            }
            result[lin+height*col]= x_1+x_2;
        }
    }
}


double getNormI(double * a, double * b)
{
    double temp;
    double m_rel=0;
    for(int i=0; i<N; i++)
    {
        temp=sqrt(a[i]*a[i] + b[i]*b[i]);
        if(temp > m_rel)
        {
            m_rel=temp;
        }
    }
    return m_rel;
}

void IterationTV(double * Im, double tau , double theta,
                 double sigma ,double lambda , int itterations ,double * mask, double* result
{
    //function x_1 = TV_inpainting( Im,  tau , theta, sigma ,
    //lambda , N ,mask)
    double * x= malloc(N*sizeof(double));//x=Im;
    double * x_bar = malloc(N*sizeof(double));//x_bar=x;

    double * y1= malloc(N*sizeof(double));//y1=zeros(size(Im));
    double * y2= malloc(N*sizeof(double));//y2=zeros(size(Im));
    double * t1= malloc(N*sizeof(double));
    double * t2= malloc(N*sizeof(double));
    double * t = malloc(N*sizeof(double));
    double * x_1 = malloc(N*sizeof(double));
    double * inter = malloc(N*sizeof(double));
    double ** g= malloc(2*sizeof(double*));
```

```
g[0] = malloc(N*sizeof(double));
g[1] = malloc(N*sizeof(double));

for(int pix=0; pix<N; pix++)
{
    x[pix]=Im[pix];
    x_bar[pix]=Im[pix];
    t[pix]=Im[pix];
    x_1[pix]=Im[pix];
}

double normi;
double normalizer;

for(int i=0; i<itterations; i++) //for i=1:N
{
    gradient(x_bar,g);//[a,b]=gradient_chambolle(x_bar);

    for(int pix=0; pix<N; pix++)
    {
        t1[pix]= y1[pix] - sigma * g[0][pix];//t1=y1-sigma*a;//********
        t2[pix]= y2[pix] - sigma * g[1][pix];//t2=y2-sigma*b;//********
    }

    normi= getNormI(t1, t2);
    if(1>normi)
    {
        normalizer=1;
    }
    else
    {
        normalizer=normi;
    }

    for(int pix=0; pix<N; pix++)
    {
        y1[pix]=t1[pix]/normalizer;//y1=t1/max(1,norm);
        y2[pix]=t2[pix]/normalizer;//y2=t2/max(1,norm);
        g[0][pix]=y1[pix];
        g[1][pix]=y2[pix] ;
    }

    divergence( g,inter);

    //t=x-tau * divergence_chambolle(y1,y2);
    for(int pix=0; pix<N; pix++)
    {
        t[pix]=x[pix]-tau*inter[pix];
    }

    //x_1=(ones(size(mask))-mask).*t +
    //mask.*(t+tau*lambda*Im)/(1+tau*lambda);
    for(int pix=0; pix<N; pix++)
    {
        if(mask[pix]==0)
        {
            x_1[pix]=t[pix];
        }
```

```
            else
            {
                x_1[pix]=(t[pix]+ tau*lambda*Im[pix])
                        /(1+tau*lambda);
            }
        }

        //x_bar = x_1 + theta*(x_1-x);
        for(int pix=0; pix<N; pix++)
        {
            x_bar[pix]= x_1[pix] + theta * (x_1[pix] - x[pix]);
        }

        //x=x_1;
        for(int pix=0; pix<N; pix++)
        {
            x[pix]=x_1[pix];
        }
    }

    for(int pix=0; pix<N; pix++)
    {
        result[pix]=x_1[pix];
    }

    /*free*/
    free(x);
    free(x_bar );
    free(y1 );
    free(y2 );
    free(t1 );
    free(t2 );
    free(t );
    free(x_1 );
    free( inter);
    free(g[1] );
    free(g[0] );
    free( g);
}


int main()
{
    /*define constants */
    char *name_input="C:\\Documents and Settings\\fabien\\Mes documents\\Stage\\donnee.txt";
    char *name_mask="C:\\Documents and Settings\\fabien\\Mes documents\\Stage\\mask.txt";
    char *name_output="C:\\Documents and Settings\\fabien\\Mes documents\\Stage\\sortie.txt";

    /*opening the file */
    FILE *input_stream = fopen (name_input, "r");
    FILE *mask_stream = fopen (name_mask, "r");
    FILE *output_stream = fopen (name_output, "w");

    height = 512;
    width = 512;
    N= height*width;

    double * valeurs_in= malloc(N*sizeof(double));
```

```
    double * valeurs_out= malloc(N*sizeof(double));
    double * valeurs_mask= malloc(N*sizeof(double));

    for(int i=0; i<N; i++)
    {
        fscanf(input_stream,"%lf",&(valeurs_in[i]));
        fscanf(mask_stream,"%lf",&(valeurs_mask[i]));
    }

    fclose(input_stream);
    fclose(mask_stream);

    //double ** gradient = malloc(2*sizeof(double*));

    for(int i=0; i<N; i++)
    {
        valeurs_in[i]=valeurs_in[i]*valeurs_mask[i];
    }

    IterationTV(valeurs_in, 5, 1, 0.02, 1000,
                10000, valeurs_mask, valeurs_out);

    /*Ecriture du fichier de sortie.*/
    for(int i=0; i<N; i++)
    {
        fprintf(output_stream,"%f\n",valeurs_out[i]);
    }

    free(valeurs_in);
    free(valeurs_out);
    free(valeurs_mask);

    fclose(output_stream);
    exit(EXIT_SUCCESS);
}
```