MASTER EN SCIENCES MATHÉMATIQUES

Algorithmes de traitement d'images numériques appliqués à l'astrophotographie

Aviles-Romariz, Mathilde

Award date:
2017

Awarding institution:
Universite de Namur

Remerciements

Je souhaite remercier toutes les personnes qui m'ont aidée tout au long de ce mémoire.

Je tiens à remercier tout particulièrement Monsieur André Füzfa, mon promoteur, qui m'a accompagnée, conseillée et encouragée pour ce travail.

Je remercie également mes parents pour la relecture, ma soeur ainsi que mon entourage qui m'ont encouragée et soutenue lors de l'élaboration de ce mémoire.

Un tout grand merci à toutes et à tous.

Résumé

Ce mémoire s'intéresse aux traitements d'images numériques appliqués à l'astrophotographie. En particulier, il s'est orienté vers le recalage d'images. Il s'agit d'une méthode consistant à superposer deux photographies contenant un même objet afin de récolter plus d'informations qu'une seule image pouvait nous apporter. Ces procédés sont également très utilisées en imagerie médicale. Par ailleurs, nous expliquons les motivations qui poussent à s'intéresser à cette technique dans le domaine de l'astrophotographie (particularité de la rotation de champ par exemple).

Le document développe les différents aspects théoriques utilisés lors des applications et les raisons qui nous mènent à nous y intéresser. Par exemple, nous allons aborder les concepts de mesures de similarité (coefficient de Pearson), des méthodes d'optimisation applicables à des fonctions non différentiables (algorithme de Pattern Search). Ensuite, la deuxième partie expose les différents résultats des stratégies exploitées.

De plus, les différentes analyses apportent les avantages et inconvénients d'utiliser une méthode plutôt qu'une autre. De cette manière, nous pourrons utiliser celle qui nous convient le mieux lors d'applications à de réelles images d'astronomie. Ces photographies utilisées ont toutes été réalisées par Monsieur André Füzfa, mon promoteur.

Le mémoire se termine par un résumé des résultats obtenus ainsi que les pistes d'amélioration qui peuvent être apportées.

Mots-clés : traitement d'images numériques - algorithmes de recalage d'images - astrophotographie - méthodes d'optimisation - mesures de similarité.

Abstract

This master thesis is interested by digital image processing applied to astrophotography. In particular it is oriented toward image registration. This method consist on superimposing two photographies containing a same object in the aim to collect more informations than an unique image could bring us. Those process are also useful in medical imaging. Furthermore, we explain two motivations which prompt being interested to this technic in the astrophotography domain (particularity of field rotation for instance).

This document develops the different theoretical sides used when we apply them and the reasons that make us interested. For example, we are going to speak about concepts of similarity measures (Pearson coefficient), of methods of optimisation applicable to non differentiable functions (algorithm of Pattern Search). Then, the second part exposes the different results of exploited strategies.

Moreover, the different analyses bring some advantages and disadvantage to use a method rather than another. In this way, we will able to use which suit us when we will apply them for real images of astronomy. Those photographies are all made by Mr André Füzfa, my supervisor.

The master thesis end with an abstract of results and improvement tracks that we could bring.

Keywords : digital image processing - image registration algorithm - astrophotography - optimisation methods - similarity measures.

Table des matières

Introduction

Ce mémoire s'intéresse aux différents traitements d'images numériques appliqués à l'astrophotographie. En effet, cette dernière est une discipline de l'astronomie qui permet de photographier des objets du ciel ; il n'est pas nécessaire que celui-ci soit nocturne pour en faire. Il est possible de photographier des objets planétaires ainsi que des éléments du ciel profond tels que des nébuleuses ou des galaxies. Ces deux types d'objets seront à considérer de manières distinctes si nous souhaitons les photographier. En effet, il s'agira du temps de pose qui changera ; cela permet d'obtenir davantage d'informations sur la photographie.

Il faut néanmoins comprendre ce mécanisme (Legault [7]) ; cette question du temps de pose. Le principe premier de la photographie est la capture des photons, ceux-ci ayant un comportement aléatoire. Ce sont les photosites qui permettent de les capturer et de stocker des électrons. Ils sont capables de stocker aussi longtemps que nous voulons des électrons libérés par effet photoélectrique. Par conséquent, lancer une pose sur un appareil numérique consiste à laisser les capteurs exposés à la lumière pour qu'ils puissent libérer une quantité d'électrons vers les photosites. Après cela, ces derniers sont vidés de leur stock d'électrons, et un convertisseur analogique-numérique mesure la quantité d'électrons pour finalement traduire en un langage compréhensible par l'ordinateur.

Cependant, la qualité d'une photographie est influencée par plusieurs facteurs d'origines diverses ; par exemple, le phénomène de signal thermique. En effet, à température ambiante, les atomes de la matière du photosite sont sujets à une agitation rapide : cela implique chez certains la libération d'électrons. Ces derniers vont alors s'ajouter aux autres électrons lors de poses plus longues ; cela crée des points plus ou moins brillants sur la photographie.

Il faut pourtant être vigilant à la rotation de la Terre lorsque nous réalisons des photographies de longues poses ; nous sommes confrontés à la rotation de la Terre. Il s'agit de la rotation de champ que nous pouvons compenser en utilisant deux outils. Le premier est une monture équatoriale motorisée qui doit être alignée parallèlement à l'axe de rotation de la Terre

afin de compenser cette dernière par un seul moteur. Le second outil est l'autoguidage ; une caméra posée sur une lunette attachée au télescope. Elle va suivre une étoile choisie (l'étoile-guide) et envoyer les corrections de mouvements à la monture avec des algorithmes de contrôle. Pourtant, malgré ces précautions, une rotation de champ résiduelle persiste. D'une part, cela est dû au fait que l'alignement et le suivi du moteur ne sont pas parfaits. D'autre part, bien que l'autoguidage compense bien cet effet, il ne se limite qu'à suivre une seule étoile ; plus nous nous en éloignons, plus l'effet de la rotation de champ résiduelle s'amplifie.

Après cette étape, viennent les méthodes de traitements d'images numériques. C'est à cela que s'intéresse ce mémoire ; il se divise en trois parties. Les fonctions, les codes sont uniquement réalisés ou utilisés via le logiciel MATLAB.

La première partie consiste en une initiation aux traitements d'images : les objets manipulés, les différentes fonctions existantes, ... Une méthode de traitement d'images est par exemple la déconvolution. Il s'agit d'un processus consistant à enlever l'effet de « flou » dans l'image ; il en existe d'ailleurs plusieurs méthodes dont l'algorithme de Lucy-Richardson. Cet effet de flou peut être produit par exemple par la buée environnante.

La seconde partie aborde, de manière théorique, le centre de la discussion : le recalage d'images. En effet, cette méthode permet d'enlever les inconvénients des longues poses. De fait, lorsque nous prenons des images du ciel profond, nous nous confrontons à la rotation de la Terre : la rotation de champ abordé dans le paragraphe précédent. Si le suivi n'est pas adéquat, les étoiles (censées être des objets ponctuels) deviendront des traits. Une manière alternative est de réaliser une série de poses courtes ; c'est là qu'intervient le recalage d'images notamment. Grâce à cette méthode, nous pourrons superposer adéquatement les différentes photographies et récolter les informations comme si nous avions une pose plus longue.

Le second avantage du recalage d'images en astrophotographie est qu'il permet de soustraire des éléments qui polluent les images de poses courtes. Ces éléments peuvent être un passage d'un avion, d'un satellite, d'un nuage ou d'un coup de vent qui aurait déplacé le télescope. En effet, les longues poses prennent environ 15 à 20 minutes pour capturer les photons. Le fait d'obtenir une image dégradée après ce temps de patience peut irriter l'astrophotographe ; heureusement le recalage d'images permet d'enlever ces défauts.

Un inconvénient de cette méthode est que nous exploitons des images à courtes poses ; ces dernières possèdent un rapport signal sur bruit faible

rendant le recalage d'images plus difficile à réaliser.

Le recalage d'images implique alors la considération d'une mesure de (di-) similarité que nous souhaitons (minimiser) maximiser. Nous nous pencherons sur les méthodes de recalage basées sur l'optimisation. Par conséquent, nous analyserons les différentes méthodes d'optimisation existantes qui peuvent nous intéresser potentiellement ainsi qu'une série de mesures.

La troisième et dernière partie consiste à appliquer différentes combinaisons de méthodes d'optimisation avec des mesures de similarité à des photographies. Dans un premier temps, les images contiendront un objet « simple » (comme un logo) où nous connaissons les paramètres de recalage. Ensuite, nous appliquons les meilleurs candidats à de réelles photographies d'objets astronomiques réalisées par mon promoteur, Monsieur André Füzfa.

Première partie

Introduction aux traitements d'images numériques

Chapitre 1

Introduction

Dans cette première partie du mémoire, nous nous intéresserons aux techniques de traitement d'images numériques appliquées à des photographies d'objets astronomiques tels que la Lune, le Soleil, les nébuleuses, les galaxies,... Plus particulièrement, nous nous intéresserons aux algorithmes d'amélioration d'images numériques appliqués à l'astrophotographie.

En effet, lors de la prise de la photo, celle-ci pourrait avoir été plus ou moins dégradée :

- soit par les turbulences atmosphériques : par exemple la buée ;
- soit par une erreur de guidance du téléscope : cela modifie la forme ponctuelle d'une étoile et laisse, pour certains cas, apparaître une forme ovale de l'objet ;
- soit par le bruit : il est essentiellement thermique. En effet, les cellules du capteur renvoient un signal de bruit aléatoire dont le niveau est proportionnel à la température.

Avant de parler de ce qu'est le traitement d'imagerie numérique, il est intéressant de savoir d'où provient ce concept, de connaître son origine historique. En réalité, ce n'est pas une idée aussi ancienne que peuvent être la réflexion à propos du système solaire ou bien l'introduction du concept de fonction. En effet, la première photographie a été réalisée par Joseph Nicéphore Niépce en 1826. À cette époque, il avait inventé avec son frère, Claude, un appareil qui permettait de capturer les photons.

Il a fallu attendre les années 60 pour qu'à la fois, la photographie numérique soit inventée ainsi que les techniques de traitement d'images numériques qui l'accompagnent. En effet, ces dernières étaient intéressantes pour l'imagerie médicale, l'imagerie satellite ou simplement l'amélioration des photographies. Cependant, leur coût était beaucoup trop élevé avec les outils informatiques qui étaient mis à disposition à cette époque.

Heureusement, dans les années 70, les ordinateurs étaient moins chers et les logiciels dédiés à ce type de traitement étaient prêts.

À partir des années 2000, le traitement d'images numériques devient l'outil le plus commun dans ce domaine (également parce que cela devient moins coûteux)[28]. Nous voyons donc bien qu'il s'agit d'un domaine récent et très utilisé.

Ce premier chapitre va aborder les notions de bases qui surviennent lorsque nous parlons de traitement d'images numériques comme le type RGB ou bien l'histogramme d'une telle image.

Ensuite, nous distinguerons deux types de domaines où nous pourrons appliquer les traitements que sont le domaine spatial et le domaine fréquentiel qui ont chacun leurs particularités. Nous introduirons le concept de déconvolution, qui permettra notamment d'enlever le bruit ou autre dégradation de l'image ; ainsi que différents algorithmes qui sont en lien, notamment celui de Lucy-Richardson.

Chapitre 2

Notions de base

2.1 Convention et représentation d'une image numérique

Une image f est représentée par une matrice de pixels (x, y) de dimension $m \times n$ où x et y sont les coordonnées spatiales et l'amplitude de f à n'importe quelle coordonnée (x, y) est appelée *intensité* de l'image en ce point. Plus la valeur de $f(x, y)$ sera élevée, plus l'intensité augmentera. Lorsque x, y et les valeurs d'amplitude de f sont tous des quantités discrètes, nous appelons l'image, une *image numérique*. Ainsi, nous pouvons constater qu'une telle image est composée d'un nombre fini d'éléments qui sont appelés *pixels*.

Par convention, le premier élément $(0, 0)$ de la matrice f est le pixel se situant sur le coin supérieur gauche. Le schéma à gauche dans la figure 2.1 illustre cela.

Remarque 2.1.1. *En MATLAB, le premier point de coordonnées de la matrice f est l'élément se situant au point* $(1, 1)$.

Concernant les valeurs que peut prendre une image, il existe différents moyens de les représenter : une image peut appartenir à différentes *classes*.

Par exemple, pour une image de format TIFF ou JPEG, les classes utilisées sont souvent `uint8` (entiers non signés de 8 bits dans l'intervalle $[0, 255]$) et `logical` (les valeurs sont soit 0, soit 1). Ces classes utilisent 1 byte pour représenter chaque pixel. La classe utilisée dépend du but de l'image. Par exemple, dans le cas de l'imagerie médicale, les classes `uint16` (entiers non signés de 16 bits dans l'intervalle $[0, 65535]$) et `int16` (entiers signés de 16 bits dans l'intervalle $[-32768, 32767]$) sont les plus employées. Ces dernières classes utilisent 2 bytes par élément.

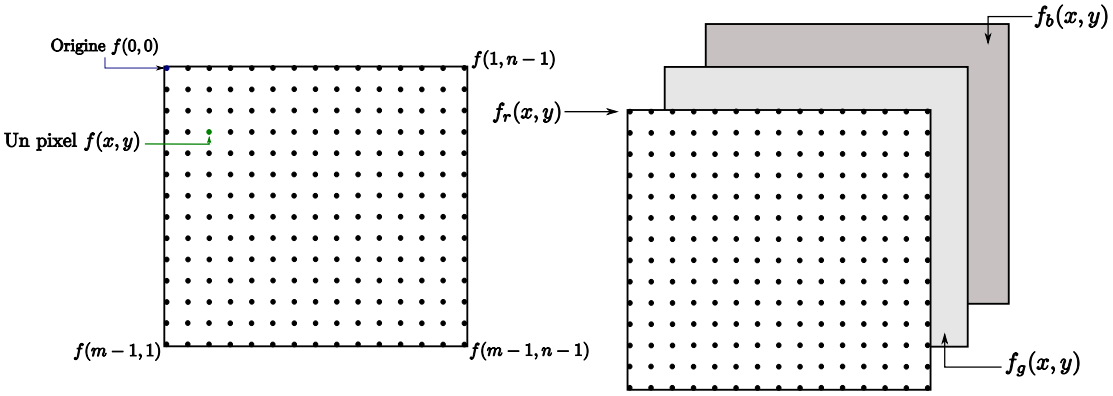


FIGURE 2.1 – Représentation schématique d'une image $f(x,y)$ à gauche et ses différentes couches monochromes à droite dans le cas d'une image de type RGB

2.2 Système RGB

Les images numériques peuvent être de différents types tels que

- les images à niveaux de gris ;
- les images binaires ;
- les images indexées ;
- les images RGB.

Souvent, nous rencontrons des images colorées et nous allons nous concentrer sur les images de type RGB.

Une *image de type RGB* est composée de trois images monochromes individuelles c'est-à-dire que l'image est une matrice de taille $m \times n \times 3$ où chaque « couche » définit l'intensité des couleurs rouge, vert et bleu. Nous noterons la couche de la couleur rouge par $f_r(x,y)$, celle de la couleur verte par $f_g(x,y)$ et enfin celle de la couleur bleue par $f_b(x,y)$. Le schéma à droite de la figure 2.1 montre ce type d'image.

Remarque 2.2.1. *Si une image RGB est de classe* `double`*, l'intervalle de valeurs est* $[0,1]$*, si elle est de classe* `uint8` *, l'intervalle est* $[0,255]$*, etc.*

2.3 Histogramme

L'*histogramme* d'une image numérique avec L niveaux d'intensité total dans l'intervalle $[0,G]$ est défini par une fonction discrète

$$h(r_k) = n_k \ ,$$

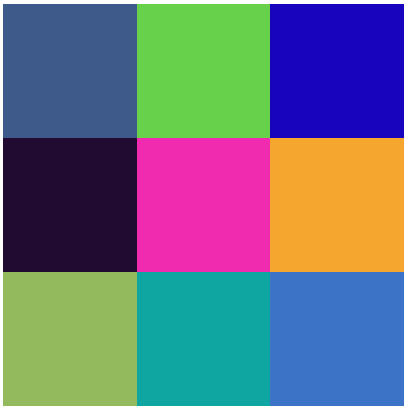


FIGURE 2.2 – Exemple d'une matrice de taille $3 \times 3 \times 3$

où r_k est la k-ème intensité dans l'intervalle $[0, G]$ et n_k est le nombre de pixels dans l'image dont le niveau d'intensité est r_k. Selon le type de données que contient l'image, nous avons plusieurs valeurs de G qui peuvent survenir :

- une image de classe `uint8` aura $G = 255$;
- une image de classe `uint16` aura $G = 65535$.

Il est parfois intéressant de connaître l'histogramme d'une image pour comprendre l'effet d'un filtre par exemple.

2.4 Exemple

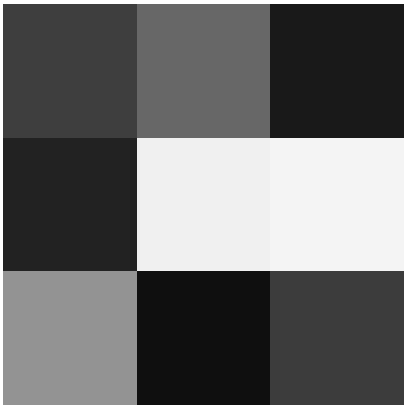
Afin d'illustrer ces premières notions, voici une image de type RGB : une matrice de taille $3 \times 3 \times 3$ (figure 2.2).

Les données qui suivent correspondent aux différentes couches

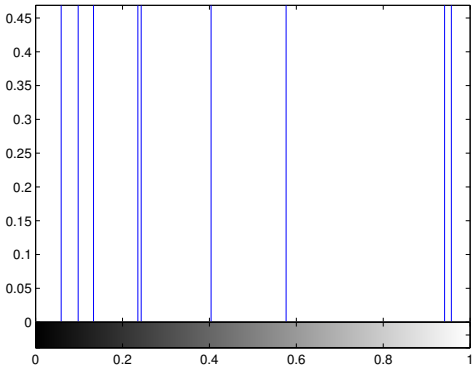
$$f_r = \begin{bmatrix} 0.2417 & 0.4039 & 0.0965 \\ 0.1320 & 0.9421 & 0.9561 \\ 0.5752 & 0.0598 & 0.2348 \end{bmatrix}, \quad f_g = \begin{bmatrix} 0.3532 & 0.8212 & 0.0154 \\ 0.0430 & 0.1690 & 0.6491 \\ 0.7317 & 0.6477 & 0.4509 \end{bmatrix}$$

$$\text{et} \quad f_b = \begin{bmatrix} 0.5470 & 0.2963 & 0.7447 \\ 0.1890 & 0.6868 & 0.1835 \\ 0.3685 & 0.6256 & 0.7802 \end{bmatrix}.$$

L'image est de classe `double` c'est-à-dire ses valeurs sont des nombres à virgule flottante, double précision dans l'intervalle $[-10^{308}, 10^{308}]$. Les figures 2.3, 2.4 et 2.5 représentent la couche correspondante à gauche c'est-à-dire (a) et l'histogramme à droite (b).

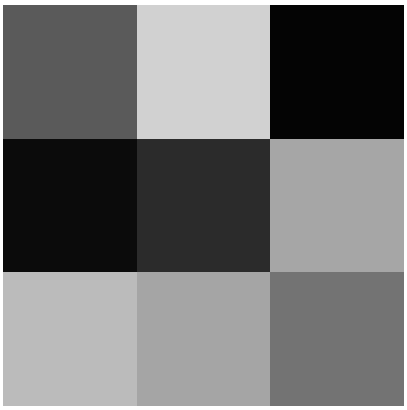


(a) Image

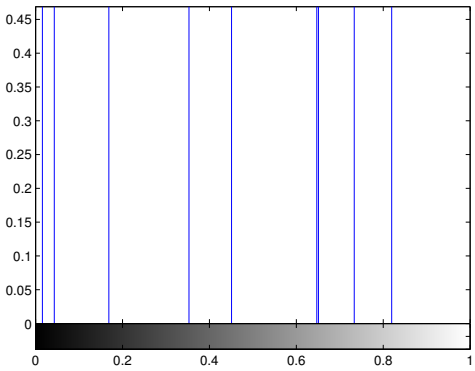


(b) Histogramme

FIGURE 2.3 – Composante rouge de l'image f_r

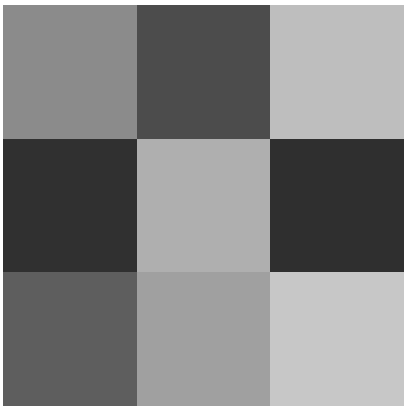


(a) Image

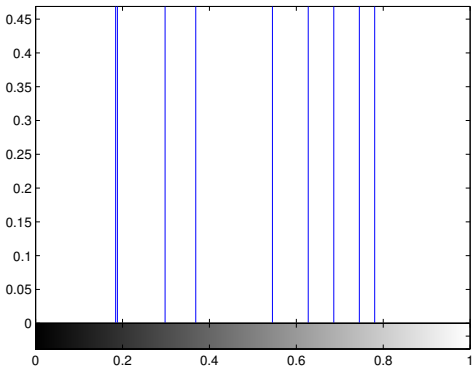


(b) Histogramme

FIGURE 2.4 – Composante verte de l'image f_g



(a) Image



(b) Histogramme

FIGURE 2.5 – Composante bleue de l'image f_b

Pour mieux comprendre les concepts précédents, concentrons-nous sur la couche rouge de l'image représentée à la figure 2.3. Intuitivement, en regardant l'image avec les trois couches, en particulier le pixel $f(2,2)$ (élément central de l'image), nous pouvons déduire que la couleur rose est probablement dûe à un niveau élevé d'intensité de couleur rouge. La figure 2.3 confirme bien cela puisque le pixel correspondant $f_r(2,2) = 0.9421$ est presque blanc et l'histogramme l'illustre également (avant-dernière droite verticale).

De plus, en analysant les valeurs des trois couches, nous pouvons voir qu'elles sont toutes distinctes, ce qui explique le fait que le nombre de pixels pour chaque niveau d'intensité est égal dans chaque histogramme.

Chapitre 3

Domaine spatial

Il faut savoir qu'il existe différentes manières de réaliser des transformations sur une image :

- soit en utilisant le *domaine spatial* qui se réfère à l'image elle-même, c'est-à-dire que les méthodes utilisées dans ce cas sont basées sur la manipulation directe des pixels de l'image ;
- soit en utilisant le *domaine fréquentiel* en utilisant la transformée de Fourier qui offre une bonne flexibilité dans l'esthétique et l'implémentation dans les résultats des filtres tels que l'amélioration de l'image, la restauration, ...

3.1 Filtrage spatial

Le *filtrage spatial* est une catégorie de traitement sur le domaine spatial. Il repose sur une procédure composée de plusieurs étapes :

1. Sélectionner un point (x, y) ;
2. Appliquer une opération qui n'implique que les pixels dans un voisinage prédéfini d'un élément (x, y) ;
3. Le résultat de cette opération se situe en ce point (x, y) considéré ;
4. Répéter les étapes précédentes pour tous les pixels de l'image.

De plus, le filtrage est caractérisé par *linéaire* ou *non linéaire* selon que l'opération sur les pixels l'est ou pas. Dans la catégorie de filtrage linéaire, il existe la corrélation et la convolution qui seront développées dans le point suivant.

3.2 Distinction Corrélation - Convolution

3.2.1 Définition

Considérons une image f. La *corrélation* est un processus qui consiste à, pour chaque point de l'image f, y centrer une matrice w (de préférence de taille impaire par facilité). Ensuite, la réponse de cette opération est la somme des produits des coefficients et des pixels du voisinage du point correspondant. Cette matrice considérée s'appelle un *filtre* ou un *masque*. Supposons que le masque w soit de taille $m \times n$ impaire. Nous pouvons résumer l'opération décrite ci-dessus par

$$w(x,y) \circ f(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i,j) f(x+i, y+j),$$

où $a = (m-1)/2$ et $b = (n-1)/2$. Ces derniers éléments désignent donc l'étendue du filtre puisque nous prendrons l'information sur les a (respectivement b) voisins du pixel central dans la direction x (respectivement y) pour construire l'image.

Ce qui nous intéresse est la *convolution* qui n'est rien d'autre que le même processus que la corrélation alors que le filtre w est tourné de 180 degrés et que nous noterons w'. Nous pouvons remarquer que $w'(x,y) = w(-x,-y)$. En conséquence de cela, l'opération de convolution peut s'écrire de différentes manières

$$\begin{aligned} w(x,y) * f(x,y) &= \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(-i,-j) f(x+i, y+j) \\ &= \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i,j) f(x-i, y-j). \end{aligned}$$

La figure 3.1 représente cette opération.

3.2.2 Exemple

Prenons la photographie du Soleil et son histogramme repris dans la figure 3.2.

En réalité, il existe plusieurs façons d'utiliser la convolution. En effet, le filtre w peut être défini de différentes manières et donne donc différents résultats. Pour illustrer cela, prenons deux filtres w distincts, w_1 et w_2 où le premier est une matrice carrée de taille 3, composée uniquement de 1 tandis que le deuxième filtre est rectangulaire procédant à une moyenne. Ici, il s'agit d'une matrice carrée de taille 50 et contenant par conséquent 50^2 éléments. Ainsi pour procéder à une moyenne sur les 50^2 éléments, il nous faut diviser le tout par

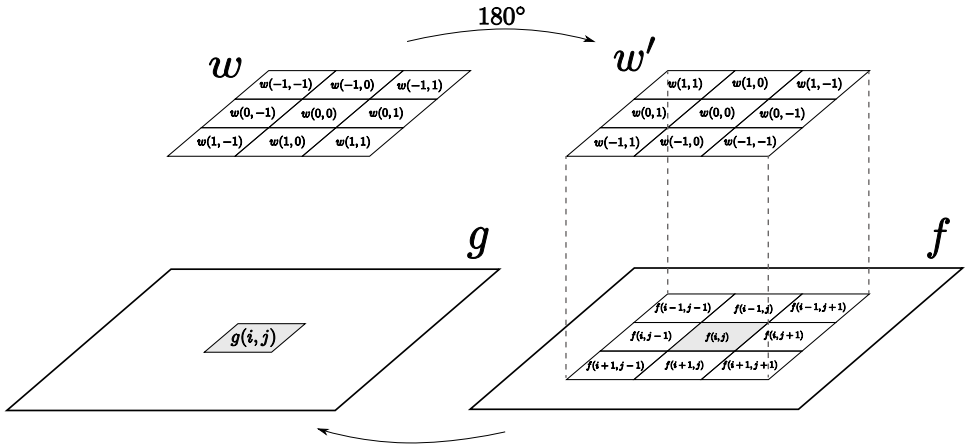


FIGURE 3.1 – Schéma du processus de convolution utilisant par exemple un filtre w de taille 3×3

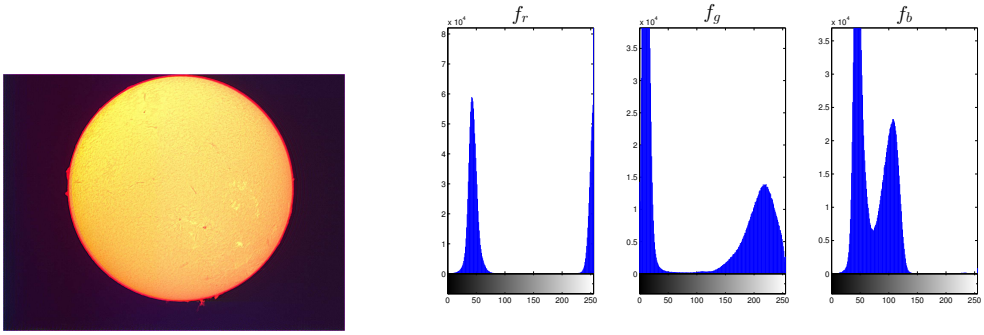


FIGURE 3.2 – Photographie du Soleil et son histogramme

le nombre d'éléments, soit multiplier par $50^{-2} = 0.4 \times 10^{-3}$. Par conséquent, les deux filtres sont définis de la manière suivante

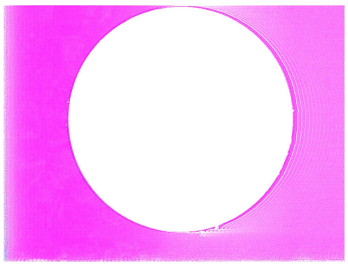
$$w_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{et} \quad w_2 = \begin{bmatrix} 0.4 \times 10^{-3} & \dots & 0.4 \times 10^{-3} \\ \vdots & \ddots & \vdots \\ 0.4 \times 10^{-3} & \dots & 0.4 \times 10^{-3} \end{bmatrix}.$$

En notant g_1 et g_2 le résultat de la convolution avec w_1 et w_2 respectivement nous avons

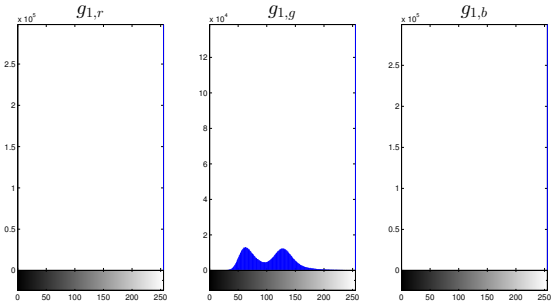
$$g_1(x, y) = w_1(x, y) * f(x, y) \quad \text{et} \quad g_2(x, y) = w_2(x, y) * f(x, y).$$

Les figures 3.3 et 3.4 montrent les différents effets ainsi que les histogrammes respectifs.

Nous pouvons remarquer que pour le premier filtre w_1, l'intensité des couleurs change complètement. En effet, étant donné que le filtre va simplement additionner les valeurs d'un certain voisinage, l'intensité du pixel fixé va croître vers la valeur 255 (soit le blanc) et ce, pour les trois couches monochromes.



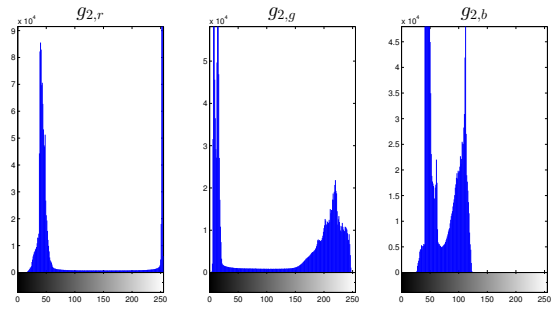
(a) Résultat $g_1(x, y)$



(b) Histogramme

FIGURE 3.3 – Résultats du processus de convolution en utilisant filtre w_1

(a) Résultat $g_2(x, y)$



(b) Histogramme

FIGURE 3.4 – Résultats du processus de convolution en utilisant filtre w_2

En regardant l'histogramme de l'image originale, le taux de pixels en intensité proche de zéros dans la couche verte est élevé contrairement aux deux autres ; ce qui explique la présence de pixels au centre dans le graphe de $g_{1,g}$ dans la figure 3.3.

Pour le second filtre w_2, les traits deviennent plus flous. En effet, comme son nom l'indique, il va procéder à une moyenne et donc nous pouvons dire intuitivement que les traits seront de moins en moins définis étant donné que la valeur de chaque intensité deviendra moyennement proche. D'ailleurs dans l'histogramme se trouvant dans la figure 3.4, nous pouvons comparer avec celui de la figure 3.2 en constatant que les taux de pixels se condensent légèrement de manière horizontale. L'effet d'une moyenne est donc de lisser une image mais ne permet pas de faire ressortir les détails fins.

Chapitre 4

Domaine fréquentiel

4.1 Définition

Comme dit précédemment, le domaine fréquentiel se basera sur la transformée de Fourier.

Soit $f(x, y)$ une image numérique de taille $m \times n$ avec $x = 0, 1, ..., m-1$ et $y = 0, 1, ..., n-1$. *La transformée de Fourier discrète* à deux dimensions de $f(x, y)$, notée $F(x, y)$, est donnée par l'équation

$$F(u, v) = \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} f(x, y) e^{-j2\pi\left(\frac{ux}{m} + \frac{vy}{n}\right)},$$

avec $u = 0, 1, ..., m-1$, $v = 0, 1, ..., n-1$ et nous utilisons la convention d'ingénieur $j^2 = -1$.

Remarque 4.1.1. *Il est possible d'exprimer l'exponentielle en termes de cosinus et de sinus afin d'analyser leurs fréquences avec les variables u et v*

$$F(u, v) = \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} f(x, y) \left[\cos\left(2\pi\left(\frac{ux}{m} + \frac{vy}{n}\right)\right) - j \sin\left(2\pi\left(\frac{ux}{m} + \frac{vy}{n}\right)\right)\right].$$

Ainsi le domaine fréquentiel est le système de coordonnées où l'image est engendrée par $F(u, v)$ avec u et v les variables fréquentielles. Nous pouvons ainsi faire l'analogie avec le domaine spatial qui est le système de coordonnées où l'image est donnée par $f(x, y)$ avec x et y les variables spatiales.

L'inverse de la transformée de Fourier discrète est donné par

$$f(x, y) = \frac{1}{mn} \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} F(u, v) e^{j2\pi\left(\frac{ux}{m} + \frac{vy}{n}\right)},$$

avec $x = 0, 1, ..., m-1$ et $y = 0, 1, ..., n-1$.

La transformée de Fourier d'une image réelle donne une matrice de complexes, et donc deux matrices réelles. C'est pourquoi la méthode principale pour analyser la transformée visuellement est de calculer le *spectre* c'est-à-dire le module de $F(u, v)$, qui est une fonction réelle contrairement à $F(u, v)$.

Considérons $R(u, v)$ et $I(u, v)$ les composantes réelles et imaginaires de $F(u, v)$. Le spectre de Fourier est défini par

$$|F(u, v)| = \left(R^2(u, v) + I^2(u, v)\right)^{\frac{1}{2}}.$$

Une autre manière équivalente est d'analyser le spectre de puissance défini par

$$P(u, v) = |F(u, v)|^2 = R^2(u, v) + I^2(u, v).$$

L'angle de phase de la transformée est défini par

$$\phi(u, v) = \arctan\left(\frac{I(u, v)}{R(u, v)}\right).$$

Nous pouvons utiliser ces deux fonctions pour exprimer $F(u, v)$ sous la forme polaire

$$F(u, v) = |F(u, v)|e^{j\phi(u,v)}.$$

Le spectre est un tableau où les composantes déterminent les intensités d'une image et la phase correspondante est un tableau d'angles qui apporte une information sur la localisation des objets dans l'image.

4.2 Filtrage dans le domaine fréquentiel

Le théorème suivant est important et intéressant dans ce qui va suivre dans cette section.

Théorème de convolution :

$$f(x, y) * h(x, y) \Leftrightarrow H(u, v)F(u, v),$$

et inversément,

$$f(x, y)h(x, y) \Leftrightarrow H(u, v) * F(u, v),$$

où la fonction $H(u, v)$ se réfère à la *fonction de transfert* du filtre. Comme pour les filtres spatiaux, nous pourrons choisir une fonction de transfert qui modifie $F(u, v)$ d'une manière spécifique.

4.2.1 Filtre passe-bas (lissage)

Précédemment, nous venons juste de mentionner la fonction de transfert d'un filtre ; il s'agit d'une notion qui permet de définir un filtre *passe-bas*. En effet, c'est lorsque la fonction de tranfert atténue les composantes à hautes fréquences (c'est-à-dire à grandes valeurs de (u, v) de l'image $F(u, v)$) tandis qu'elle laisse les basses fréquences relativement inchangées lorsqu'elle est multipliée par $F(u, v)$. Le résultat d'un tel filtre est une image floue, lisse.

Il existe différents types de filtres passe-bas dont les suivants :

- *filtre passe-bas idéal* : possède la fonction de transfert suivante

$$H(u,v) = \begin{cases} 1 & \text{si } D(u,v) \leq D_0 \\ 0 & \text{si } D(u,v) > D_0, \end{cases}$$

  où D_0 est un nombre positif et $D(u, v)$ est la distance du point (u, v) par rapport au centre du filtre.

  Remarque 4.2.1. *Le lieu du point $D(u, v) = D_0$ est un cercle. Par conséquent, puisque le filtre sera multiplié par la transformée de Fourier d'une image, nous voyons que le filtre idéal multiplie par 0 les composantes de $F(u, v)$ en dehors du cercle et laisse inchangées toutes les composantes se trouvant sur et dans le cercle (elles sont multipliées par 1). En d'autres termes, nous allons supprimer toutes les fréquences supérieures à D_0 à partir d'un couple de fréquences données (centre (u, v) du cercle dans le domaine fréquentiel).*

- *filtre passe-bas Butterworth d'ordre k* : la fonction de transfert est donnée par

$$H(u,v) = \frac{1}{1 + \left(\frac{D(u,v)}{D_0}\right)^{2k}}.$$

- *filtre passe-bas Gaussien* : possède la fonction de transfert suivante

$$H(u,v) = e^{-\frac{D^2(u,v)}{2\sigma^2}}.$$

4.2.2 Filtre passe-haut (rend plus net)

Il s'agit d'une procédure opposée à celle des filtres passe-bas. En effet, au lieu d'avoir un résultat plus lisse (flou), nous obtenons un résultat plus net en atténuant les basses fréquences et en laissant les hautes fréquences relativement inchangées de la transformée de Fourier. En d'autres termes, cela peut s'exprimer en fonction de la fonction de transfert d'un filtre passe-bas.

Considérons la fonction de transfert $H_{LP}(u,v)$ d'un filtre passe-bas, la fonction de transfert du *filtre passe-haut* correspondant est donnée par

$$H_{HP}(u,v) = 1 - H_{LP}(u,v).$$

Comme pour les filtres passe-bas, il existe plusieurs approches de filtres passe-haut :

- *filtre passe-haut idéal* : possède la fonction de transfert suivante

$$H(u,v) = \begin{cases} 0 & \text{si } D(u,v) \leq D_0 \\ 1 & \text{si } D(u,v) > D_0, \end{cases}$$

  où D_0 est un nombre positif de « coupure » de fréquence et $D(u,v)$ est la distance du point (u,v) par rapport au centre du filtre.

- *filtre passe-haut Butterworth d'ordre k* : la fonction de transfert est donnée par

$$H(u,v) = \frac{1}{1 + \left(\frac{D_0}{D(u,v)}\right)^{2k}}.$$

  En effet,

$$\begin{aligned} H(u,v) &= 1 - \frac{1}{1 + \left(\frac{D(u,v)}{D_0}\right)^{2k}} \\ &= \frac{1 + \left(\frac{D(u,v)}{D_0}\right)^{2k} - 1}{1 + \left(\frac{D(u,v)}{D_0}\right)^{2k}} \\ &= \frac{\left(\frac{D(u,v)}{D_0}\right)^{2k}}{1 + \left(\frac{D(u,v)}{D_0}\right)^{2k}} \\ &= \frac{\left(\frac{D(u,v)}{D_0}\right)^{2k}}{\left(\frac{D(u,v)}{D_0}\right)^{2k}\left[\left(\frac{D_0}{D(u,v)}\right)^{2k} + 1\right]} \\ &= \frac{1}{1 + \left(\frac{D_0}{D(u,v)}\right)^{2k}}. \end{aligned}$$

- *filtre passe-haut Gaussien* : possède la fonction de transfert suivante

$$H(u,v) = 1 - e^{-\frac{D^2(u,v)}{2\sigma^2}}.$$

Chapitre 5

Point Spread Function (PSF)

5.1 Définition

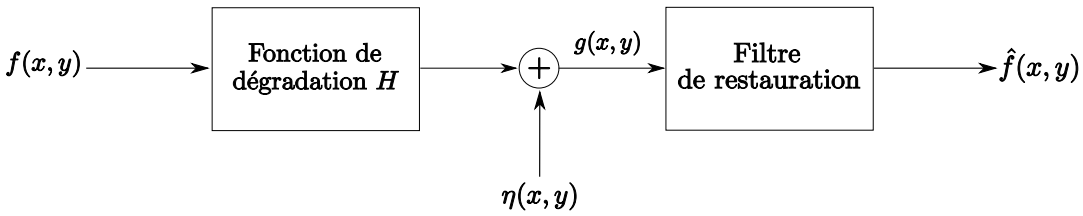


FIGURE 5.1 – Schéma du processus de Dégradation - Restauration

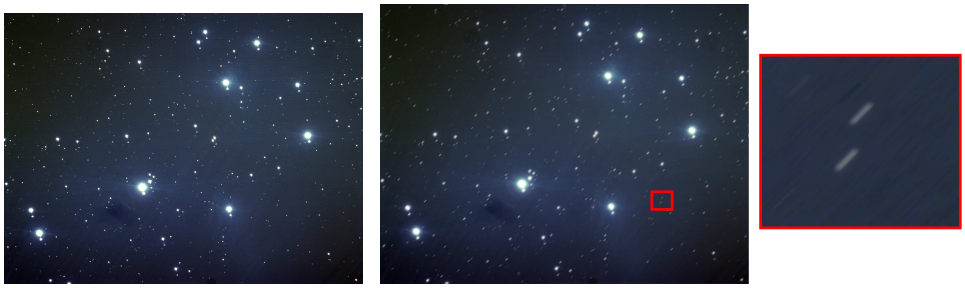
La figure 5.1 schématise les processus de Dégradation - Restauration d'une image. D'ailleurs, il est possible de modéliser cela en considérant simplement une image dégradée, notée $g(x, y)$, qui est le résultat de la dégradation de l'image originale $f(x, y)$. Cette dégradation peut être notée par $H[f(x, y)]$ en ajoutant du bruit, noté $\eta(x, y)$. Nous obtenons donc la formulation suivante

$$g(x, y) = H[f(x, y)] + \eta(x, y).$$

En astronomie, la fonction H est dûe aux limitations des télescopes. Par exemple, l'image d'une étoile obtenue par un télescope même idéal n'est pas un point mais une tâche lumineuse appelée tâche d'Airy qui est dûe à la diffraction de la lumière dans l'instrument. D'autres limitations proviennent des défauts de l'instrument : défauts de polissage des lentilles, abérrations de chromatisme ou de sphéricité, etc. Tout ceci entre dans la fonction H.

Ensuite, le bruit est dû aux sources extérieures : perturbations atmosphériques, thermiques, bruit du capteur, etc.

L'objectif de la restauration est de trouver une estimation $\hat{f}(x, y)$ de



(a) Originale (b) En appliquant le PSF

FIGURE 5.2 – Photographie de M45, les Pléiades

l'image originale $f(x, y)$. Par conséquent, plus nous connaissons H et $\eta(x, y)$, plus $\hat{f}(x, y)$ sera proche de $f(x, y)$. De plus, si H est une fonction linéaire invariante spatiallement, il est possible de montrer dans le domaine spatial que

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y),$$

où la fonction $h(x, y)$ est la représentation spatiale de la fonction de dégradation et se réfère au *Point Spread Function (PSF)*. Il s'agit donc d'un terme qui vient de la considération de la dégradation de l'image originale. De plus, $h(x, y)$ est également la réponse impulsionnelle du système (l'image d'une source parfaitement ponctuelle).

Il est possible de faire le parallèle avec le domaine fréquentiel. En effet, en utilisant le théorème de convolution, nous pouvons aisément écrire la relation suivante

$$G(u, v) = H(u, v)F(u, v) + N(u, v).$$

5.1.1 Exemple

Il est possible d'approximer les effets d'un mouvement linéaire provenant de l'appareil photo d'un certain nombre `LEN` de pixels et d'un angle `THETA` exprimé en degrés

```
PSF = fspecial ('motion', LEN, THETA)
```

Par exemple, prenons `LEN`= 25 et `THETA`= 45. Nous pouvons aisément constater dans la figure 5.2 la simulation de déplacement des pixels avec un angle de 45 degrés.

Chapitre 6

Bruit

Le *bruit* d'une image est la présence d'informations parasites qui s'ajoutent de façon aléatoire dans les détails d'une photographie numérique. En effet, il est toujours présent durant l'acquisition, la transmission et les étapes de traitement.

Il est nécessaire de distinguer les concepts de signal et de bruit. Le *signal* c'est ce qui nous intéresse scientifiquement dans les données tandis que le bruit ne l'est pas. Par conséquent, séparer le bruit du signal est évidemment une bonne chose. De plus, si nous pouvions estimer le bruit, avec la connaissance des propriétés instrumentales, la restauration d'image deviendrait plus simple.

Le bruit est souvent pris comme suivant une loi de Poisson (lié à l'arrivée des photons) et/ou une loi Gaussienne [21].

Il existe différents types de bruits dont les suivants :

- *bruit Gaussien* : les origines principales d'un tel bruit proviennent de sources naturelles ; par exemple : il serait causé par une pauvre luminosité ou bien une température élevée ;

- *bruit sel et poivre* : il s'agit d'une dégradation de l'image sous la forme de pixels noirs et blancs répartis aléatoirement. Ce bruit est dû soit à des erreurs de transmission de données, ou à la défaillance d'éléments du capteur CCD, ou encore à la présence de particules fines sur le capteur d'images ;

- *bruit de grenaille* : il se produit lorsque le nombre fini de particules transportant l'énergie est suffisamment faible pour donner lieu à des fluctuations statistiques perceptibles. Un tel type de bruit suit une loi de Poisson. Rappelons que ces bruits sont aussi valables pour toute photo

FIGURE 6.1 – Photographie de la galaxie d'Andromède et zoom pour montrer le bruit présent

numérique, donc notamment avec les capteurs CMOS des appareils photos, GSM, etc ;

- *bruit de quantification* : chaque photosite d'un capteur fournit un signal analogique auquel le convertisseur analogique-numérique attribue une valeur approximative qui est la plus proche de sa valeur réelle. Cette approximation est l'erreur de quantification d'un signal. Sur l'ensemble des photosites d'un capteur, l'accumulation des erreurs de quantification génère un bruit de quantification dont la distribution est uniforme [30].

La figure 6.1 montre un exemple de présence de bruit dû en partie à la faible luminosité de la photographie et au bruit thermique du capteur (appelé le « dark »). Nous pouvons ainsi constater que le bruit est dû au capteur puisque nous voyons des pixels rouges, verts ou bleus aléatoires et indépendants. En effet, ce sont différents photosites du capteur qui ont été excités par la température et pas par les photons du ciel.

Chapitre 7

Déconvolution

Il s'agit d'un processus permettant d'enlever les effets atmosphériques c'est-à-dire de corriger les imperfections de l'image dûes par exemple à la buée, au mauvais déplacement de l'appareil photographique lors de l'acquisition,... Comme annoncé dans la section 5, considérons une image observée g d'une image originale f ayant été dégradée par un PSF h et où du bruit a été ajouté η. Rappelons l'équation modélisant cela

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y).$$

Le but est donc de trouver $f(x, y)$ connaissant $g(x, y)$ et $h(x, y)$. En effet, dans le problème de déconvolution, le PSF est supposé connu. En pratique, ce dernier est construit à partir des données ou à partir d'un modèle optique du télescope.

7.1 Méthode de Bayes

Il s'agit d'une méthode qui utilise le théorème de Bayes qui s'énonce de la manière suivante

Théorème de Bayes :

Soient A et B deux événements alors

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{7.1}$$

Ensuite, elle maximise la partie droite de l'équation (7.1). Ainsi la solution du maximum de vraisemblance va maximiser la densité $P(B|A)$ en A étant donné que la dérivée de $P(B)$ en A sera négligeable [21].

7.2 Blind deconvolution

7.2.1 Définition

L'un des problèmes le plus difficile dans la restauration d'image est d'obtenir une estimation adaptée du PSF pour utiliser les algorithmes de restauration comme celui de Lucy-Richardson. Les méthodes qui ne sont pas basées sur la connaissance spécifique du PSF sont appelées des algorithmes *blind deconvolution*.

Une approche fondamentale est basée sur l'estimateur du maximum de vraisemblance ; une stratégie d'optimisation utilisée pour obtenir des estimations de quantités falsifiées par le bruit aléatoire. Une interprétation de l'estimateur du maximum de vraisemblance est de penser à l'image donnée comme des quantités aléatoires ayant une certaine probabilité d'être produites par une famille d'autres quantités aléatoires. La fonction de probabilité est exprimée en termes de $g(x, y)$, $f(x, y)$ et $h(x, y)$ et le problème devient alors de trouver le maximum de la fonction de probabilité [5].

7.2.2 Fonction MATLAB

La fonction implémentée dans MATLAB s'écrit de la manière suivante

```
[f, PSF] = deconvblind(g, INITPSF)
```

où

- `g` est l'image dégradée ;
- `INITPSF` est une estimation initiale du PSF ;
- `PSF` est l'estimation finale du PSF ;
- `f` est l'image restaurée en utilisant le PSF estimé.

L'algorithme utilisé pour obtenir l'image restaurée est celui de Lucy-Richardson décrit dans le point suivant.

7.3 Algorithme de Lucy-Richardson

Il s'agit d'une méthode itérative non linéaire qui a été développée indépendamment par William Richardson (1972) et Leon Lucy (1974). L'algorithme suppose la connaissance du PSF (normalisé) et que la statistique de l'image peut être modélisée par une loi de Poisson. En fait, cet algorithme provient de la formulation du maximum de vraisemblance et de l'utilisation de la méthode de Bayes. De plus, puisqu'il s'agit d'une méthode itérative, il faudra à moment

ou à un autre définir un critère d'arrêt. Ce dernier peut se baser sur l'image résultante et s'arrêter si la qualité est raisonnable ou bien définir un autre critère.

Dans notre cas, nous nous intéressons à $P(g|f)$ qui s'interprète comme la probabilité d'observer une image g donnée sachant que l'image originale est f. Selon le théorème de Bayes, nous avons

$$P(f|g) = \frac{P(g|f)P(f)}{P(g)}.$$

Ce sera donc le terme $P(g)$ qui sera négligeable dans le sens où $P(g)$ est constant par rapport à f et donc n'interviendra pas pour déterminer le maximum de vraisemblance.

7.3.1 Description de l'algorithme

Comme g suit une loi de Poisson par hypothèse et a comme moyenne

$$E[g] = h * f,$$

nous avons que la probabilité $P(g|f)$ est

$$P(g|f) = \prod_{x,y} \frac{((h*f)(x,y))^{g(x,y)} e^{-(h*f)(x,y)}}{g(x,y)!}.$$

Remarque 7.3.1. *Nous manipulons des images numériques, ce qui signifie que nous considérons des images à valeurs entières (voir la section 2.1).*

Le maximum de vraisemblance est calculé en prenant la dérivée du logarithme c'est-à-dire

$$\begin{aligned}
\frac{\partial \ln(P(g|f))}{\partial f} &= \frac{\partial}{\partial f}\left[\ln\left(\prod_{x,y} \frac{((h*f)(x,y))^{g(x,y)} e^{-(h*f)(x,y)}}{g(x,y)!}\right)\right] \\
&= \frac{\partial}{\partial f}\left[\sum_{x,y} g(x,y)\ln(h*f)(x,y) - (h*f)(x,y) - \ln(g(x,y)!)\right] \\
&= \sum_{x,y} \frac{g(x,y)}{(h*f)(x,y)} * h(-x,-y) - h(x,y) \\
&= \frac{g(x,y)}{(h*f)(x,y)} * h(-x,-y) - 1,
\end{aligned}$$

où nous avons supposé que le PSF h est normalisé c'est-à-dire $\sum_{x,y} h(x,y) = 1$. De plus, nous avons utilisé la définition d'une dérivée d'un produit de convolution qui est la suivante

$$(f*g)' = f' * g = f * g'.$$

Nous pouvons également remarquer que le passage de la deuxième à la troisième ligne, le caractère négligeable de $P(g)$ que nous avions souligné précédemment : $P(g)$ est une constante par rapport à f.

Afin de trouver le maximum, il nous reste à annuler la dérivée

$$\begin{aligned}\frac{\partial \ln(P(g|f))}{\partial f} = 0 \quad &\Leftrightarrow \quad \frac{g(x,y)}{(h*f)(x,y)} * h(-x,-y) - 1 = 0 \\ &\Leftrightarrow \quad \frac{g(x,y)}{(h*f)(x,y)} * h(-x,-y) = 1.\end{aligned}$$

En multipliant de chaque côté par $f(x,y)$, nous obtenons l'équation suivante

$$\left[\frac{g(x,y)}{(h*f)(x,y)} * h(-x,-y)\right] f(x,y) = f(x,y).$$

En utilisant l'itération de Picard [21], nous pouvons avoir la solution de l'équation précédente en utilisant la suite suivante

$$f_{n+1}(x,y) = \left[\frac{g(x,y)}{(h*f_n)(x,y)} * h(-x,-y)\right] f_n(x,y).$$

7.3.2 Fonction MATLAB

Il existe une fonction déjà implémentée dans MATLAB réalisant l'algorithme de déconvolution de Lucy-Richardson décrite de la manière suivante

```
f = deconvlucy(g,PSF, NUMIT, DAMPAR, WEIGHT)
```

où

- `f` est l'image restaurée ;
- `g` est l'image dégradée ;
- `PSF` est le PSF ;
- `NUMIT` est le nombre d'itération (par défaut 10) ;
- `DAMPAR` est un scalaire (par défaut 0) qui définit un seuil de déviation de l'image résultante de l'image g : les itérations sont supprimées pour les pixels qui dévient d'une valeur `DAMPAR`de leur valeur originale. Cela supprime la génération de bruit dans de tels pixels préservant ainsi les détails de l'image ;
- `WEIGHT` est un tableau (par défaut unitaire) de même taille que `g` et assigne un poids à tous les pixels pour refléter leur qualité.

Deuxième partie

Aspects théoriques

Chapitre 1

Recalage d'images

Ce chapitre s'intéressera au recalage d'images proprement dit. D'abord en élaborant les motivations de ce traitement d'images ; ensuite en définissant ce concept ainsi qu'en détaillant les différentes étapes. Nous allons particulièrement nous concentrer sur certaines mesures de similarité qui seront utilisées dans les applications dans la seconde partie de ce mémoire. Ce chapitre s'est donc inspiré du livre de Ardeshir Goshtasby [1].

1.1 Motivations

Il s'agit d'une procédure qui permet de trouver des correspondances entre tous les points dans deux images d'un même objet. Cette correspondance est requise dans la reconstruction d'image 3-D, dans la détection et reconnaissance d'objet, dans la fusion d'images, dans l'analyse de mouvement,... Elle a de nombreuses applications notamment dans le cadre de l'imagerie médicale ou bien dans la création de panoramas. La figure 1.1 en donne un exemple concernant des photos aériennes.

En astronomie, elle permet également d'améliorer le rapport signal sur bruit. Comme l'intensité lumineuse est faible, nous pouvons additionner une série d'images pour simuler l'effet d'une pose plus longue. Or, pour additionner, il faut que les structures de l'image (par exemples les étoiles) soient bien aux mêmes endroits.

En effet, si nous ne recalons pas les images dans le cas du domaine de l'astrophotographie, nous risquons d'être confrontés au phénomène de la rotation de champ. Ce dernier provient du fait que la Terre tourne, et si nous souhaitons prendre une pose plus longue, le suivi doit être correct pour obtenir des objets bien déterminés comme des étoiles bien ponctuelles. Dans le cas où nous ne

FIGURE 1.1 – Exemple de recalage de photographies aériennes [1].

souhaiterions pas réaliser une longue pose, comme signalé au paragraphe précédent, nous pouvons simuler en réalisant plusieurs courtes poses. Si les images ne sont pas recalées et que nous les superposons toutes, nous obtenons l'image présentée à la figure 1.2.

FIGURE 1.2 – Huit prises de la Nébuleuse du Coeur superposées à des instants différents.

Remarque 1.1.1. *L'image 1.2 a été modifiée grâce au logiciel Graphic Converter 9 au niveau des contrastes afin d'avoir une meilleure visualisation de l'effet de rotation de champ dans l'image. De plus, nous pouvons constater certaines étoiles qui ne sont pas influencées par cet effet : l'une d'entre elles est l'étoile-guide ; cela survient souvent lorsque nous souhaitons photographier un objet du ciel profond. D'ailleurs, T. Legault [7] nous indique que l'amplitude du tracé est proportionnelle à la distance angulaire à l'étoile-guide. Si celle-ci*

se situe en dehors de la photographie, l'amplitude est plus accentuée. La figure 1.3 illustre cette remarque.

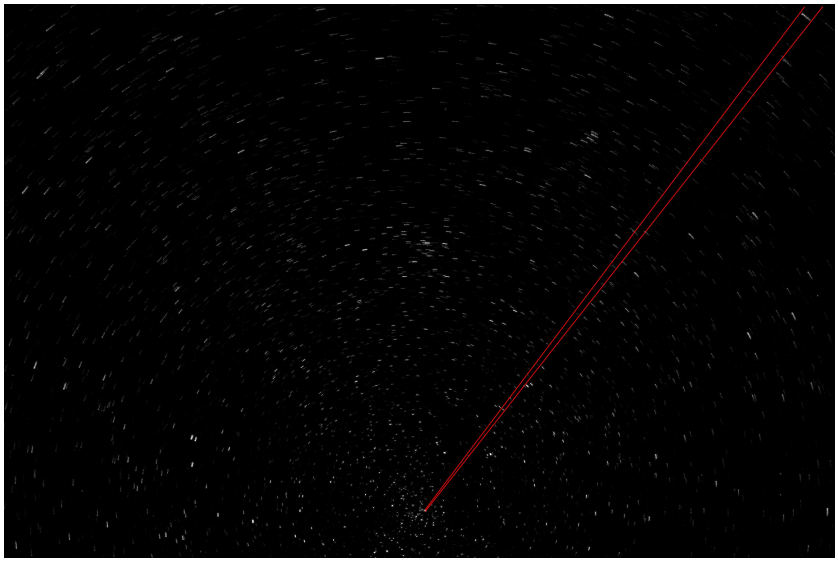


FIGURE 1.3 – Effet de la rotation de champ sur la Nébuleuse du Coeur.

Nous pouvons concevoir alors la raison d'intégrer le recalage d'images dans le domaine de l'astrophotographie. La section suivante définit le recalage d'images ainsi que les différentes étapes qui le composent.

1.2 Définition

Distinguons l'image de référence et l'image de détection. L'*image de référence* est celle que nous allons garder inchangée, tandis que l'*image de détection* est celle que nous allons transformer pour avoir la géométrie et les coordonnées spatiales de l'image de référence.

Un système de recalage d'images peut être vu comme une boîte noire qui reçoit une image de référence, une image de détection et d'autres échantillons de l'image de détection pour les aligner spatialement avec l'image de référence. Cette opération assigne les mêmes coordonnées aux points correspondants dans les images.

Une première manière de procéder, et ce lorsque les images peuvent être traitées comme des objets rigides, nous translatons et tournons simplement l'image de détection pour faire correspondre avec l'image de référence. Cependant, ce procédé n'est en général pas suffisant, ni efficace. En effet, il n'est limité qu'au recalage où il ne nécessite que de translater ou de tourner l'image de détection. Or ce n'est pas toujours le cas.

Une autre manière, plus robuste, est de détecter un certain nombre de points de contrôle dans l'image de référence et ensuite calculer les caractéristiques décrivant le voisinage de ces points ; celles qui donnent le plus d'informations sont choisies. Lorsque nous avons trouvé un ensemble de correspondances correctes dans l'image, les paramètres d'une fonction non linéaire sont déterminés afin de transformer l'espace géométrique de l'image de détection en celui de l'image de référence. La figure 1.4 résume les différentes étapes du recalage d'images.

1.2.1 Principales étapes

Mesure de similarité

Considérons deux suites de mesures $X = \{x_i : i = 1, ...n\}$ et $Y = \{y_i : i = 1, ..., n\}$ (représentent deux images) où les x_i et y_i sont les intensités des pixels dans les images correspondantes. La *similarité* entre elles est une mesure qui quantifie la dépendance entre les images.

Remarque 1.2.1. *Les deux images (les deux suites d'éléments) sont de tailles identiques.*

Une mesure de similarité est considérée comme une métrique ; si elle produit une valeur plus grande lorsque la dépendance entre les valeurs correspondantes des deux images augmente. En effet, une telle mesure S satisfait les propriétés suivantes :

1. (Distance limitée) $S(X, Y) \leq S_0$;
2. (Réflexivité) $S(X, Y) = S_0 \Leftrightarrow X = Y$;
3. (Symétrie) $S(X, Y) = S(Y, X)$;
4. (Inégalité triangulaire) $S(X, Y)S(Y, Z) \leq (S(X, Y) + S(Y, Z))\, S(X, Z)$.

où S_0 est la plus grande mesure de similarité que l'on peut trouver entre les éléments de X et Y. Il est possible de choisir la mesure de similarité en fonction des caractéristiques de l'image.

Au fil des années, plusieurs mesures de similarité ont fait leur apparition :

- le coefficient de corrélation de Pearson ;
- la mesure Tanimoto ;
- le rho de Spearman ;
- le tau de Kendall ;
- ...

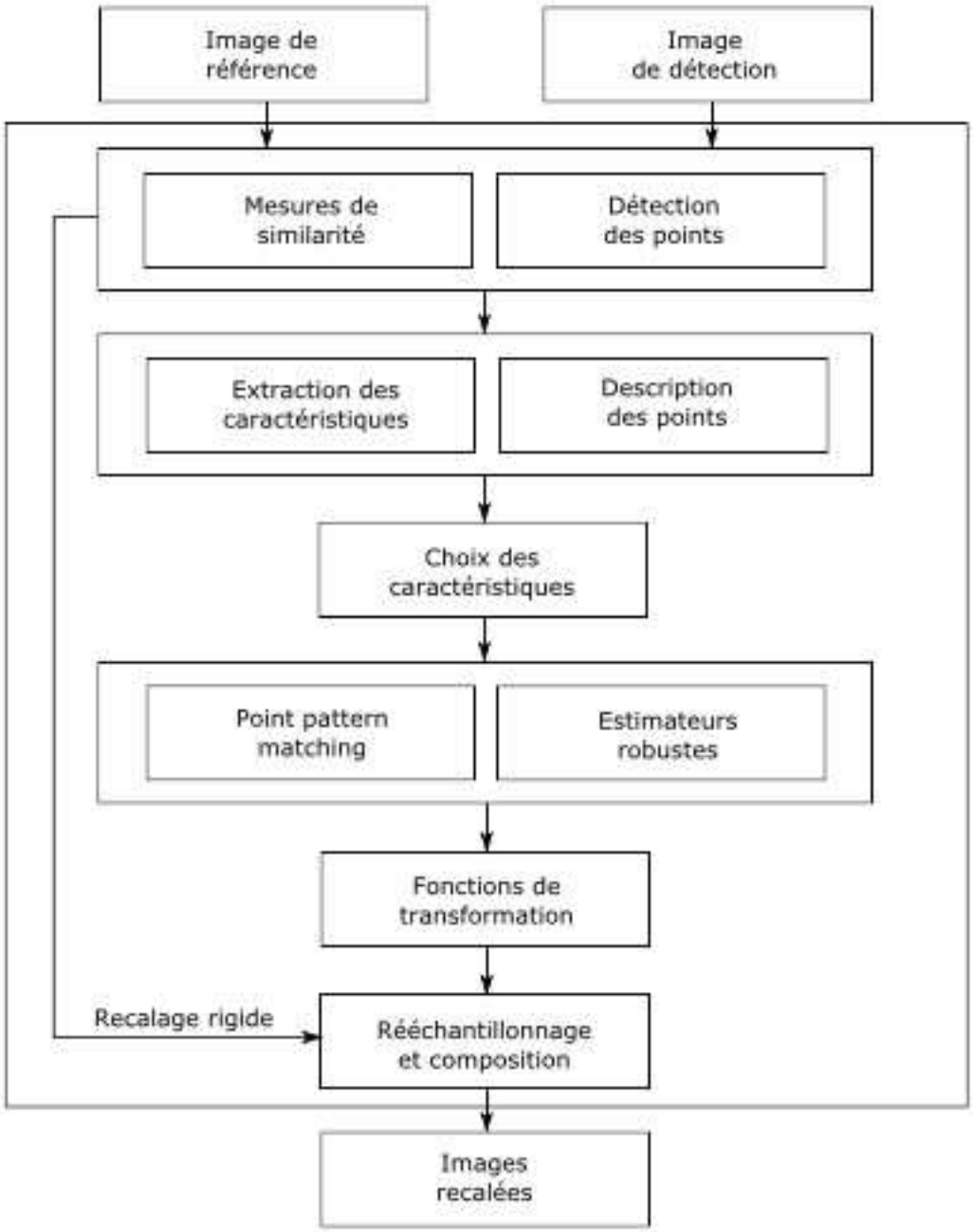


FIGURE 1.4 – Schéma des étapes du recalage d'images [1].

Détection de points

Les points caractéristiques dans une image apportent énormément d'informations sur la structure de l'objet et sont très utilisés pour l'analyse de l'image. Il est important que les points détectés soient indépendants du bruit, du flou, du contraste et des changements géométriques de sorte que les mêmes points peuvent être obtenus dans des images du même objet sous différentes

conditions environnementales.

Différents types de détecteurs de points existent :

- détecteurs basés sur la corrélation ;
- détecteurs basés sur le gradient ;
- détecteurs basés sur l'unicité ;
- détecteurs basés sur le laplacien ;
- ...

Remarque 1.2.2. *Il s'agit bien de faire de la reconnaissance de formes dans des images ; les points pour l'astronomie correspondent aux étoiles.*

Extraction des caractéristiques

Une *caractéristique* est une mesure d'une propriété globale ou locale d'une image révélant une information statistique, algébrique, géométrique, spatiale, différentielle et spectrale sur une image ou sous-image. Le problème sera donc de choisir la caractéristique qui apportera le plus d'informations sur l'image.

Pour qu'une caractéristique soit utile, elle doit produire la même valeur des images d'un même objet tant qu'elle produit des valeurs différentes d'images de différents objets.

Description des points

Un *descripteur* d'image est un vecteur caractéristique contenant diverses informations sur une image. Il s'agit d'une représentation compacte d'une image pour la distinguer d'une autre. Il est utilisé pour déterminer la correspondance entre des points de contrôle dans deux images. En effet, une fois que les points de contrôle sont détectés, des fenêtres sont choisies ayant pour centre ces points. Les informations dans ces fenêtres sont alors utilisés pour établir une correspondance entre les points des images.

Lorsque la rotation et le changement d'échelle sont inconnus, il est nécessaire de considérer des descripteurs de rotation et de changement d'échelle pour trouver les correspondances.

Il existe différents descripteurs de rotation et de changement d'échelle :

- les descripteurs basés sur l'histogramme ;
- les descripteurs basés sur le filtrage ;
- les descripteurs basés sur la rotation ;
- ...

Choix des caractéristiques

Considérons un ensemble de D caractéristiques d'une image. Le problème du choix de caractéristique est de sélectionner $d < D$ des caractéristiques qui contiennent le plus d'informations. Nous choisirons les caractéristiques de sorte que nous réalisions l'erreur de correspondance la plus petite.

Il y a trois types d'algorithmes :

- Algorithme *filter* : il choisit les caractéristiques sans tenir compte de la correspondance finale ; il enlève les caractéristiques qui sont hautement dépendants et retient celles qui sont indépendantes ;

- Algorithme *wrapper* : les caractéristiques sont choisies en tenant compte de la correspondance finale. Il est généralement le plus précis par rapport à l'algorithme filter mais coûte plus cher en opérations ;

- Algorithme hybride : il combine les méthodes précédentes.

Point pattern matching

Les points de contrôle dans une image peuvent avoir des caractéristiques associées dûes aux différences d'intensité et géométriques ainsi qu'à la présence du bruit, ce qui les rend peu précises.

Les algorithmes de *point pattern matching* utilisent la contrainte géométrique qui garde la correspondance entre les points des images pour distinguer les correspondances correctes des incorrectes. La contrainte géométrique dépend de la géométrie de l'image et celle de l'objet. Par exemple, comme les images d'un objet plat sont reliées par une transformation de projection, la contrainte de projection est utilisée pour trouver les correspondances.

Estimateurs robustes

Les méthodes robustes utilisent les correspondances des points de contrôle pour déterminer les paramètres d'une fonction de transformation. Le but des estimateurs robustes est d'identifier certaines ou toutes les correspondances correctes et utiliser leurs coordonnées pour déterminer les paramètres de transformation.

Fonctions de transformation

Une fonction de transformation utilise les coordonnées des points de contrôle correspondants dans les deux images afin d'estimer la relation géométrique

entre elles. Elle sera alors utilisée pour transformer la géométrie de l'image de détection de sorte qu'elle soit alignée spatialement avec l'image de référence.

Rééchantillonnage et composition

- Rééchantillonnage La transformation

$$\begin{cases} X = f(x, y) \\ Y = g(x, y) \end{cases} \tag{1.1}$$

  relie les coordonnées des points dans l'image de référence à celles des points correspondants dans l'image de détection. Les relations (1.1) déterminent les coordonnées (X, Y) du même point dans l'image de détection. En lisant l'intensité en (X, Y) dans l'image de détection, elle est rééchantillonnée point par point dans la géométrie de l'image de référence.

- Composition Après avoir enregistré les deux images, il faut les combiner les en une image plus grande appelée un *composite*.

  Les intensités des pixels correspondants dans la zone de chevauchement peuvent être différents à cause des différences dans les paramètres environnementaux et de capteurs durant l'acquisition de l'image (changement de luminosité à cause du temps par exemple) [1].

La section suivante propose une série de mesures de similarité dont la plupart seront analysées dans la troisième partie de ce mémoire.

1.3 Mesures de similarité étudiées

1.3.1 Le coefficient de corrélation de Pearson

Considérons les images $X = \{x_i : i = 1, ..., n\}$ et $Y = \{y_i : i = 1, ..., n\}$. Le *coefficient de corrélation de Pearson* est défini de la manière suivante :

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{[\sum_{i=1}^{n}(x_i - \bar{x})^2]^{\frac{1}{2}} [\sum_{i=1}^{n}(y_i - \bar{y})^2]^{\frac{1}{2}}}$$

où $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ et $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. De manière équivalente, en divisant le numérateur et le dénominateur par la taille des images, on obtenons l'expression

$$r = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\frac{1}{n}\left[\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2\right]^{\frac{1}{2}}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i-\bar{y})^2\right]^{\frac{1}{2}}}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{(x_i-\bar{x})}{\sigma_x}\right)\left(\frac{(y_i-\bar{y})}{\sigma_y}\right)$$

Ainsi, le coefficient de corrélation de Pearson r est compris entre -1 et 1. Lorsque r vaut 1, les images coïncident parfaitement. Tandis que lorsque r vaut -1, les images coïncident de manière parfaitement opposée.

Le fichier `pearson.m` contient les différentes instructions permettant de calculer ce coefficient :

```
1  function r = pearson(X, Y, param)
2
3  Y = geometricT(param, Y);
4
5  [n m] = size(X); % n lignes et m colonnes
6  X_moy = 0;
7  Y_moy = 0;
8  sigma_X = 0;
9  sigma_Y = 0;
10
11 for i = 1:n
12     for j = 1:m
13         X_moy = X_moy + X(i,j);
14         Y_moy = Y_moy + Y(i,j);
15     end
16 end
17
18 X_moy = X_moy / (n*m);
19 Y_moy = Y_moy / (n*m);
20
21
22 for i = 1:n
23     for j = 1:m
24         sigma_X = sigma_X + ((X(i,j) - X_moy)^2) ;
25         sigma_Y = sigma_Y + ((Y(i,j) - Y_moy)^2) ;
26     end
27 end
28
29 sigma_X = sqrt((1/(n*m)) * sigma_X);
30 sigma_Y = sqrt((1/(n*m)) * sigma_Y);
31
32 r = 0;
33
34 for i = 1:n
```

```
35      for j = 1:m
36          r = r + (((X(i,j) - X_moy)/sigma_X)*((Y(i,j) - ...
                Y_moy)/sigma_Y));
37      end
38  end
39
40  r = (1/(n*m))*r;
41  end
```

Remarque 1.3.1. *Les notations analytiques considérant les images sont sous forme d'un vecteur de taille n et non sous la forme matricielle comme considérée dans les codes. Cela ne change rien étant donné qu'alors la taille du vecteur sera $1 \times mn$ et les éléments seront multipliés avec les bonnes correspondances selon la définition.*

1.3.2 Minimum Ratio

Dans le cas où l'image de détection Y est la version bruitée de l'image de référence X et si en plus, l'amplitude du bruit est proportionnelle à la force du signal, nous avons :

$$m_r = \frac{1}{n}\sum_{i=1}^{n} r_i,$$

où

$$r_i = \min\left\{\frac{y_i}{x_i}, \frac{x_i}{y_i}\right\}.$$

Cela mesure donc la dépendance entre les pixels des différentes images. S'il n'y a pas de bruit, alors $\forall i = 1, ..., n$, $r_i = 1$. Ce qui implique que $m_r = \frac{1}{n}\sum_{i=1}^{n} 1 = 1$. De cette manière, plus les images seront similaires, plus le ratio se rapprochera de 1.

Malgré que cette mesure soit résistante au bruit, elle est sensible à la différence d'intensité entre les images. Elle possède donc son point faible lorsqu'il faut faire correspondre des images prises à des moments différents, possédant une luminosité différente ou bien prises par des appareils différents.

De plus, le calcul de cette mesure ne requiert que de simples petits calculs en chaque pixel. Par conséquent, sa complexité est de l'ordre de n.

D'ailleurs les lignes d'instructions pour calculer cette mesure sont les suivantes :

```
1  function m_r = minimumRatio(X,Y, param)
2
3      Y = geometricT(param, Y);
```

```
4
5       [m,n] = size(X);
6       r = zeros(m,n);
7       m_r = 0;
8
9       for i = 1:m
10          for j = 1:n
11              r(i,j) = min(X(i,j)/Y(i,j), Y(i,j)/X(i,j));
12              m_r = m_r + r(i,j);
13          end
14      end
15
16      m_r = m_r/(m*n);
17  end
```

1.3.3 Rho de Spearman

Dans cette mesure de similarité, on s'intéresse non pas aux intensités, mais aux rangs des images. Le *rho de Spearman* est défini par

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} \left(\mathrm{rg}(x_i) - \mathrm{rg}(y_i)\right)^2}{n(n^2-1)},$$

où $\mathrm{rg}(x_i)$ et $\mathrm{rg}(y_i)$ dénotent le rangs de x_i et y_i respectivement.

Il s'agit en réalité du coefficient de Pearson appliqué au rang de chaque image. De plus, cela permet d'être moins sensible au bruit (aux valeurs aberrantes) qui pourrait être présent.

En comparant avec la corrélation de Pearson, le rho de Spearman est moins sensible aux valeurs aberrantes et donc moins sensible au bruit. D'un point de vue calculs, ρ est plus lent à calculer que r ; cela est dû au fait d'ordonner les intensités de X et Y qui est de l'ordre de $n \log_2 n$ comparaisons (Ardeshir Goshtasby [1]).

Etant donné que nous venons de relever son avantage concernant la présence de bruit, lorsque l'image n'en contient pas, le rho de Spearman ρ n'est pas plus précis que le coefficient de corrélation de Pearson r. La mesure est donnée dans le fichier spearmanRho.m :

```
1   function rho = spearmanRho(X,Y, param)
2       Y = geometricT(param, Y);
3
4       [m,n] = size(X);
5       Y = redim(Y,[m n]);
6       [o,p] = size(Y);
```

```
7
8       X = reshape(X',m*n,1);
9       Y = reshape(Y',o*p,1);
10
11      if(size(X,1) ≠ size(Y,1))
12          vide = zeros(size(X,1));
13          for i = 1:size(X,1)
14             vide(i) = vide(i) + Y(i);
15          end
16          Y = vide;
17      end
18
19      if size(X,1)≠size(Y,1)
20          error('x and y must have equal number of rows.');
21      end
22
23      % Find the data length
24      N = length(X);
25      % Get the ranks of x
26      R = crank(X)';
27
28      for i=1:size(Y,2)
29          % Get the ranks of y
30          S = crank(Y(:,i))';
31          % Calculate the correlation coefficient
32          rho(i) = 1-6*sum((R-S).^2)/N/(N^2-1);
33      end
34  end
```

où la fonction `crank(X)` est décrite par les lignes de codes :

```
1   function r = crank(x)
2       u = unique(x);
3       [xs,z1] = sort(x);
4       [z1,z2] = sort(z1);
5       r = (1:length(x))';
6       r = r(z2);
7
8       for i = 1:length(u)
9           s = find(u(i)==x);
10          r(s,1) = mean(r(s));
11      end
12  end
```

Cette fonction permettant de calculer cette mesure provient du fichier partagé dans File Exchange du site MathWorks [9].

1.3.4 Tau de Kendall

Avant de définir ce qu'est le tau de Kendall, il est nécessaire de spécifier ce que sont des *pixels concordants* et *discordants*. Dans le cas où $\text{sign}(x_j - x_i) = \text{sign}(y_j - y_i)$, on dit que les pixels sont concordants. Par contre, lorsque $\text{sign}(x_j - x_i) = -\text{sign}(y_j - y_i)$, les pixels sont dits discordants. À présent, nous pouvons donner l'expression du *tau de Kendall*

$$\tau = \frac{N_c - N_f}{n(n-1)/2},$$

où N_c et N_d sont le nombre de paires concordantes et discordantes respectivement. Cette mesure est l'une des plus coûteuses en temps de calculs étant donné qu'elle est de l'ordre de n^2 (le coefficient de corrélation est de l'ordre de n et le rho de Spearman est de l'ordre de $n \log_2 n$). En effet, lorsque nous calculons τ nous devons effectuer $n(n-1)/2$ comparaisons (ou combinaisons).

Cependant, étant donné que nous allons travailler avec des images à hautes résolutions, nous n'allons pas retenir cette mesure dans la suite de ce mémoire. Effectivement, la complexité est plus élevée que celle des autres mesures et cela prendrait donc un temps considérable pour obtenir un résultat.

1.3.5 Structural Similarity

Cette mesure, comme d'autres, permet de mesurer la qualité visuelle d'une image compressée par rapport à l'image originale. La mesure de deux images X et Y de tailles n est définie de la façon suivante :

$$SSIM(X,Y) = \frac{(2\mu_X\mu_Y + c_1)(2\sigma_X\sigma_Y + c_2)(2\text{cov}_{XY} + c_3)}{(\mu_X^2 + \mu_Y^2 + c_1)(\sigma_X^2 + \sigma_Y^2 + c_2)(\sigma_X\sigma_Y + c_3)}, \tag{1.2}$$

où

- μ_X, μ_Y sont les moyennes des images X et Y respectivement ;
- σ_X, σ_Y sont les écarts-types des images de la même manière ;
- cov_{XY} est la covariance de X et Y ;
- $c_1 = (k_1 L)^2$ où $k_1 = 0.01$ par défaut et L représente la dynamique des valeurs des pixels (par exemple, il vaut 255 si les pixels sont codés sur 8 bits) ;
- $c_2 = (k_2 L)^2$ où $k_2 = 0.03$ par défaut ;
- souvent, $c_3 = \frac{c_2}{2}$.

Grâce à cette dernière considération, nous pouvons réécrire la formule (1.2) de la manière suivante :

$$SSIM(X,Y) = \frac{(2\mu_X\mu_Y + c_1)(2\sigma_{XY} + c_2)}{(\mu_X^2 + \mu_Y^2 + c_1)(\sigma_X^2 + \sigma_Y^2 + c_2)}.$$

La philosophie de cette mesure est la suivante : nous allons mesurer la qualité de deux images grâce au fait que nous supposons que la perception visuelle humaine est très adaptée pour extraire l'information sur la structure d'un objet. Par conséquent, elle peut donner une bonne approximation pour percevoir une déformation de l'image. Cependant, en réalité, la formule n'implique jamais la perception humaine et n'est pas aussi précise comme annoncé (Wang [25]).

Les lignes de code permettant de calculer cette mesure de similarité sont décrites ci-dessous.

```
function s = ssimMoi(X,Y, param)

Y = geometricT(param, Y);
[n m] = size(X); % n lignes et m colonnes
Y = redim(Y,[n m]);
k_1 = 0.01;
k_2 = 0.03;
L = 255;

c_1 = (k_1*L)^2;
c_2 = (k_2*L)^2;
c_3 = c_2/2;

X_moy = 0;
Y_moy = 0;
sigma_X = 0;
sigma_Y = 0;

for i = 1:n
    for j = 1:m
        X_moy = X_moy + X(i,j);
        Y_moy = Y_moy + Y(i,j);
    end
end

X_moy = X_moy / (n*m);
Y_moy = Y_moy / (n*m);


for i = 1:n
    for j = 1:m
        sigma_X = sigma_X + ((X(i,j) - X_moy)^2) ;
```

```
33          sigma_Y = sigma_Y + ((Y(i,j) - Y_moy)^2) ;
34      end
35  end
36
37  sigma_X = sqrt((1/(n*m)) * sigma_X);
38  sigma_Y = sqrt((1/(n*m)) * sigma_Y);
39
40  XY = X'*Y;
41  [o p] = size(XY);
42  XY_moy = 0;
43  for i = 1:o
44      for j = 1:p
45          XY_moy = XY_moy + XY(i,j);
46      end
47  end
48
49  XY_moy = XY_moy/(o*p);
50  sigma_XY = XY_moy - (X_moy*Y_moy);
51
52  s = ((2*X_moy * Y_moy+c_1)*(2*sigma_XY+c_2))/ ((X_moy^2 + ...
       Y_moy^2+c_1)*(sigma_X^2 + sigma_Y^2 + c_2));
53  end
```

1.3.6 Exemples d'application des différentes mesures

Dans le but de clarifier ce qui a été dit précédemment et parce que cela sera intéressant dans l'étude des différentes méthodes, le tableau 1.1 résume les différentes valeurs des mesures de similarité. Pour ce faire, considérons une image de référence, une image bruitée, une image déformée par une transformation géométrique et une image floue. Tentons de confirmer les différentes approches via ces exemples illustrés à la figure 1.5. Le tableau 1.2 donne le temps d'exécution (en secondes) de chaque calcul de mesures.

(a) Image originale. (b) Image bruitée. (c) Image avec une transformation géométrique. (d) Image floue

FIGURE 1.5 – Différentes formes du logo de l'Université de Namur.

Mesure de similarité	Image bruitée	Image transformée	Image floue
Coefficient de Pearson	0,8921	-0,2558	0,9430
Minimum Ratio	0,9695	-2,5584	0,9282
Rho de Spearman	0,9334	-0,1968	0,9214
SSIM	0,3707	0,0693	0,7803

TABLE 1.1 – Résumé des valeurs des différentes mesures de similarité.

Mesure de similarité	Complexité	Image bruitée	Image transformée	Image floue
Coefficient de Pearson	n	0,029	0,016	0,023
Minimum Ratio	n	0,017	0,016	0,016
Rho de Spearman	$n \log_2(n)$	0,279	16,614	19,311
SSIM	-	0,047	0,042	0,039

TABLE 1.2 – Résumé des différents temps d'exécution (en secondes) des mesures de similarité.

Remarque 1.3.2. *Evidemment, il existe bien d'autres mesures de similarité. Or le but du mémoire est de proposer certaines méthodes et le fait de considérer une liste plus exhaustive ferait l'objet d'un travail encore plus large.*

À présent, lorsque nous analysons le tableau 1.1, nous pouvons constater que la mesure de Minimum Ratio est moins sensible au bruit par rapport aux autres (bien qu'il n'y ait pas beaucoup de différence). Par contre, SSIM est loin derrière.

De plus, lorsque nous examinons la troisième colonne, c'est la fonction SSIM qui apporte une valeur plus grande de similarité. Cela peut s'expliquer par le fait que la philosophie de base de cette mesure est la perception visuelle humaine et par conséquent, l'oeil jugerait que les images se ressemblent relativement : il n'y a pas de bruit ni de flou qui s'est ajouté à la transformation.

Concernant l'image floue, le coefficient de Pearson a une valeur plus élevée ; mais la différence avec le second n'est de l'ordre que du centième (Minimum Ratio).

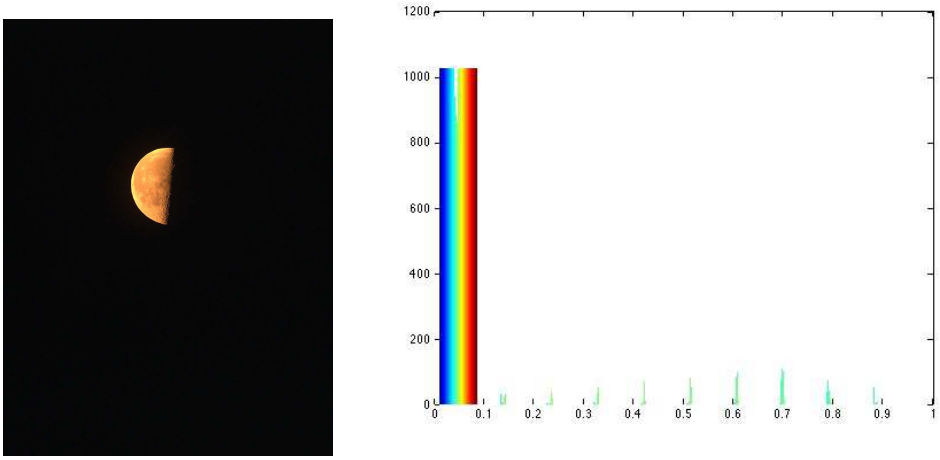
Lorsque nous regardons le tableau 1.2, le ρ de Spearman est plus coûteux en temps par rapport aux autres, nous pouvons donc sentir l'ampleur de la complexité des mesures lorsque nous allons devoir les optimiser.

La section suivante aborde une méthode de détection de points. En ef-

fet, cela est intéressant lorsque nous souhaitons diminuer le temps de calculs ou bien repérer des points caractéristiques à l'image. Plus particulièrement, nous allons introduire la transformée de Hough pour la détection de cercles.

1.4 La transformée de Hough (cercles)

L'intérêt d'aborder cette méthode provient du fait que nous souhaitons appliquer les différents algorithmes à l'astrophotographie. Nous sommes donc face soit à une image planétaire soit à une image du ciel profond. Ce dernier type d'objet ne sera pas traité de la même manière que pour une image planétaire étant donné que les caractéristiques ne sont pas identiques. Les images du ciel profond contiendront plus d'objets diffus et moins délimités par une forme particulière comme c'est le cas pour les images planétaires. De plus, l'histogramme ne sera pas du même type car les intensités lumineuse ne sont pas répartis de la même manière. Pour illustrer l'idée, considérons deux images : la lune et la nébuleuse d'Orion représentés à la figure 1.6 et 1.7.

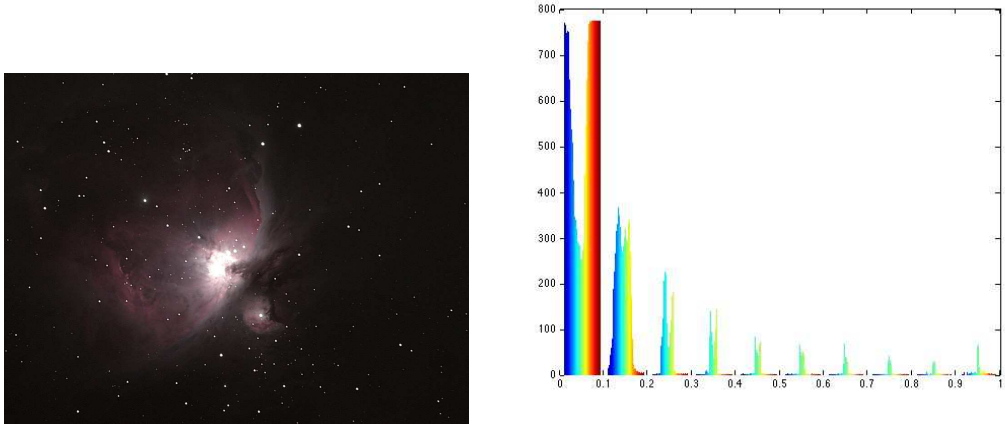


(a) Image. (b) Histogramme.

FIGURE 1.6 – Image de la Lune et l'histogramme associé.

Les histogrammes sont globalement similaires mais nous voyons malgré tout la différence au niveau de la répartition. De fait, l'image de la Lune contient peu d'intensités différentes (le fond noir et la couleur claire de la Lune) tandis que celle de la nébuleuse est plus relativement répartie.

Remarque 1.4.1. *Le fait d'avoir un histogramme multicolore provient du fait que l'image analysée est en couleurs RVB et représente donc le niveau des intensités des trois couleurs.*



(a) Image. (b) Histogramme.

FIGURE 1.7 – Image de la Nébuleuse d'Orion et l'histogramme associé.

Par conséquent, il nous faut trouver un moyen de détecter des points caractéristiques de l'image tels que des étoiles qui ont la forme d'un cercle. C'est ainsi que nous introduisons la transformée de Hough pour les cercles. En effet, elle permet de détecter ces formes dans une image et en plus, elle est robuste avec la présence de bruit.

Pour ce faire, nous devons estimer les paramètres permettant de définir un cercle : les coordonnées du centre ainsi que son rayon. Rappelons donc l'expression d'un cercle

$$(x-a)^2 + (x-b)^2 = r^2,$$

où (a, b) sont les coordonnées du centre du cercle et r est son rayon.

Les explications de l'algorithme proviennent d'un mélange de Graf [6], des fiches MATLAB [13] et de Wikipédia [27]. Celui-ci se divise en trois étapes :

1. **Calcul de la matrice d'accumulation**

   D'abord, fixons le rayon d'un cercle afin de diminuer le nombre de paramètres à estimer ; nous avons un espace de paramètres à deux dimensions. Ensuite, les pixels du premier plan ayant un gradient élevé (l'intensité ou la couleur change brusquement) sont désignés comme des candidats et peuvent donner un *vote* dans la matrice d'accumulation. Dans une transformée de Hough classique pour les cercles, les candidats votent dans la forme qui se trouve autour d'eux (un cercle avec un rayon fixe). La figure 1.8 illustre cette étape.

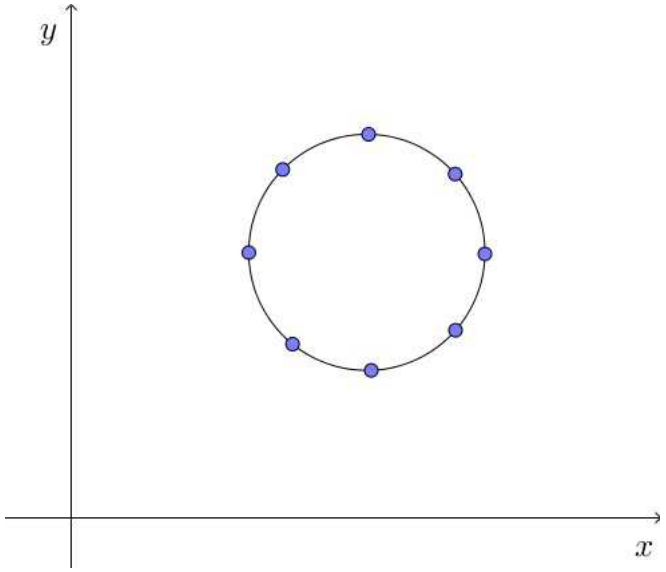


FIGURE 1.8 – Les points candidats appartenant au cercle sont mis en évidence dans l'espace d'origine (espace de l'image) [19].

Cette matrice d'accumulation permet donc de connaître les points d'intersection dans l'espace de paramètres. En réalité, les éléments de la matrice représentent le nombre de fois que passe un cercle par le point correspondant dans grille de valeurs dans l'espace des paramètres. On appelle alors ces nombres des *votes*. Au départ, ces derniers sont assignés à zéro.

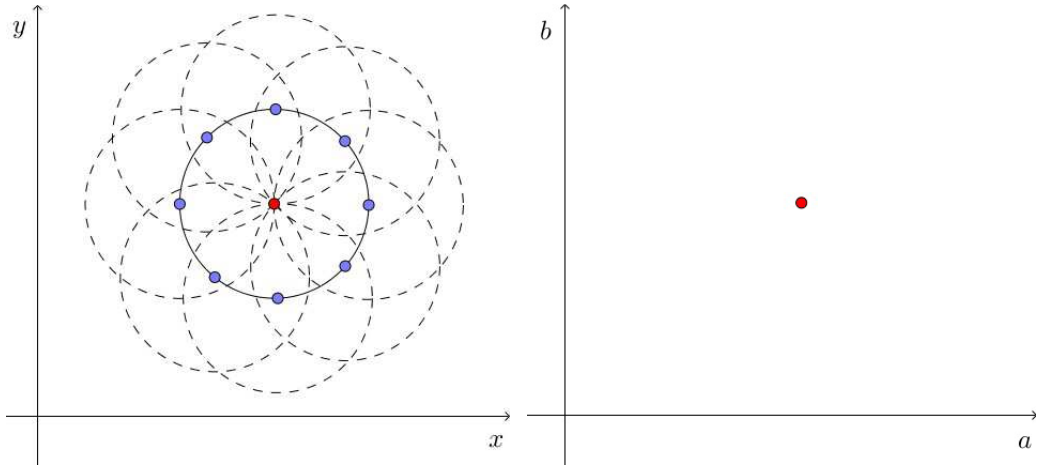
2. **Estimation du centre**

Après l'assignation de tous les votes, nous gardons le point qui en possède le plus. Les coordonnées de ce maximum sont les coordonnées du cercle dans l'espace d'origine représentées à la figure 1.9.

3. **Estimation du rayon**

Dans le cas où nous ne connaîtrions pas la valeur du rayon, nous pouvons utiliser le même processus que pour l'estimation des centres décrite précédemment (en réduisant la dimension de l'espace de paramètres).

Sinon, dans le cas où le rayon ne serait pas connu, nous nous trouvons dans un espace à trois dimensions (trois paramètres à estimer). Par conséquent, le lieu des points se trouvera sur la surface d'un cône. En effet, nous ne connaissons pas la valeur du rayon, ce qui implique que nous devons parcourir tous les cercles ayant un certain centre (fixé dans un premier temps) où le rayon varie de 0 à une valeur maximale.



(a) Intersection des cercles des candidats (b) Point retenu (résultat de l'intersection)

FIGURE 1.9 – Intersection de tous les cercles retenus pour connaître les coordonnées du cercle [19].

De cette manière, chaque point (x, y) d'un cercle de l'espace d'origine produit un cône dans l'espace de paramètres. La matrice d'accumulation ne sera plus à deux dimensions mais trois et ses éléments (qui sont à présent des triplets (a, b, r)), correspondent au nombre de fois que la surface d'un cône est passée par un point. La figure 1.10 à la page 53 représente la nouvelle situation dans l'espace des paramètres à trois dimensions.

Cette transformée existe en fonction MATLAB et s'écrit

```
[centers,radii] = imfindcircles(A,radiusRange)
```

Inputs

- `A` est l'image où il faut détecter les cercles ;
- `radiusRange` est un intervalle des rayons spécifié par `[rmin rmax]` (il est également possible de fixer une seule valeur du rayon).

Outputs

- `center` est une matrice contenant les coordonnées des cercles ;
- `radii` est un vecteur colonne contenant la valeur des rayons de chaque cercle.

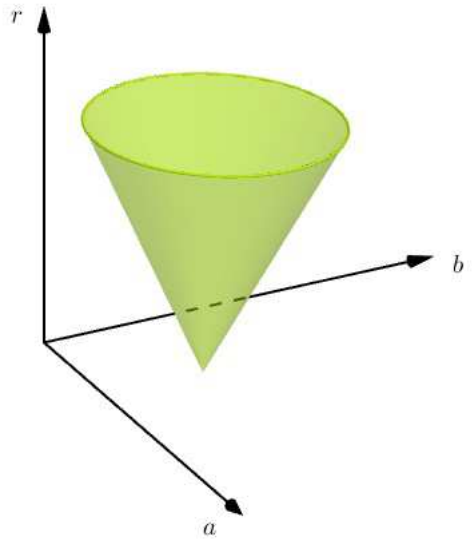


FIGURE 1.10 – Surface de possibilité de définir un cercle dans l'espace de paramètres [19].

1.5 Transformation géométrique

Dans le cadre du mémoire, nous nous intéresserons aux transformations géométriques rigides. En effet, le but est de recaler des images provenant du même appareil photographique et de récolter un maximum d'informations dans les images prises durant la même nuit, par exemple. Durant les temps de poses, l'appareil n'aura à priori pas bougé et par conséquent, nous n'aurons pas de différence de taille concernant l'objet ; la distance entre les différents objets de l'image n'aura donc pas changé. Seule une rotation ou une translation aurait pu s'y appliquer.

Considérons les coordonnées de l'image de référence X (abscisses) et Y (ordonnées) ainsi que celles de l'image de détection x (abscisses) et y (ordonnées). La transformation géométrique (rigide) que nous recherchons s'exprime de la manière suivante :

$$\begin{cases} X = x\cos(\theta) - y\sin(\theta) + h \\ Y = x\sin(\theta) + y\cos(\theta) + k \end{cases}.$$

En utilisant une formulation matricielle, nous avons

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

De cette manière, nous avons explicité l'expression de la matrice de rotation R telle que

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

ainsi que celle de translation T telle que

$$T = \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix}.$$

Ainsi, θ représente la différence de rotation mesurée dans le sens anti-horlogique entre les deux images (de déformation et de référence) ; h est le paramètre de la translation horizontale tandis que k est le paramètre de la translation verticale.

La section suivante consiste à décrire deux méthodes de recalage applicables à notre problème.

1.6 Méthodes de recalage

1.6.1 Recalage basé sur l'optimisation

On considère un recalage *optimal* lorsqu'il maximise une mesure de similarité (ou minimise une mesure de dissimilarité) entre les deux images. Pour ce faire, il faut définir une mesure de similarité (ou de dissimilarité), fixer les paramètres de départ et développer un algorithme qui recale les images. Il est possible que l'utilisateur choisisse lui-même les paramètres initiaux de manière interactive.

De cette manière, à chaque itération, les paramètres de recalage sont affinés afin d'atteindre un résultat optimal. Nous recherchons donc les paramètres qui donneront une mesure de similarité plus grande (ou une plus petite mesure de dissimilarité) à chaque étape.

Dans les sections qui suivent, on aborde les différentes méthodes d'optimisation qui sont intéressantes dans ce mémoire.

1.6.2 Méthode de multirésolution

Cette méthode consiste à créer à partir des deux images à recaler plusieurs images de tailles de plus en plus petites. En effet, les plus petites images ré-

duisent les différences géométriques et ainsi accélèrent le processus de recalage.

Les images originales, dites au niveau 0, sont diminuées de taille ; souvent d'un facteur deux. Ainsi, nous obtenons une deuxième image dite au niveau 1. On continue le même processus avec cette nouvelle image jusqu'à ce qu'on obtienne la résolution souhaitée. Le nombre de niveau peut être donné par l'utilisateur ou bien fixé automatiquement.

Ensuite, le recalage est d'abord réalisé sur les images ayant la plus basse résolution (niveau n). En réduisant suffisamment la résolution d'une image, les différences géométriques locales sont réduites afin de permettre un recalage avec une fonction de transformation globale. Cette diminution réduit le temps de calculs et simplifient le processus du fait de la réduction de point de contrôle dans les images. De cette manière, les paramètres de recalage trouvés au niveau n, sont utilisés pour recaler l'image du niveau $n-1$. On continue le processus jusqu'à ce qu'on arrive à l'image de niveau 0.

Cette méthode est souvent utile lorsque nous voulons recaler des images avec des différences géométriques locales.

Un autre avantage que possède cette méthode, est que lorsque nous réduisons la résolution, le bruit s'estompe également. Cela permettra donc de bien recaler les images par leur vraie information (le signal) plutôt que ce soit biaisé par le bruit.

La section suivante concerne la qualité de recalage. En effet, nous avons besoin d'une mesure qui permet de quantifier la qualité ; nous ne pouvons nous restreindre à la perception visuelle humaine.

1.7 Mesure de la qualité de recalage : rapport signal sur bruit

Pourquoi s'intéresser à ce concept ? À la base, le but de ce rapport est de mesurer la qualité de transmission de l'information. En effet, le bruit survient sous différentes formes notamment dans l'information sonore ou bien l'information numérique (bien distinguer dans l'information le signal du bruit). Plus ce rapport est grand, plus le bruit est négligeable et donc mieux c'est. Il s'agit en réalité du rapport de la moyenne des pixels μ du signal et de l'écart-type σ des valeurs des pixels sur un voisinage donné :

$$SNR = \frac{\mu}{\sigma}.$$

La fonction MATLAB [17] permettant de le calculer s'écrit

```
snr = (x,y)
```

où `x` est le signal et `y` est le bruit. MATLAB nous conseille d'ailleurs d'utiliser cette fonction de cette manière dans le cas où le signal entrant n'est pas sinusoïdal nécessairement et que nous avons une estimation du bruit. Or dans nos futures applications, nous n'aurons pas forcément une estimation du bruit et de plus, nous avons à disposition deux images qui ne contiennent ni le signal ni le bruit proprement dits ; les deux sont incluses. Par conséquent, nous devons recourir à une autre alternative : *Peak Signal to Noise Ratio* [16].

En effet, il s'agit également d'une mesure de distorsion utilisée en image numérique. Il se définit par

$$PSNR = 10 \log_{10} \left(\frac{d^2}{EQM} \right),$$

où d est la dynamique du signal c'est-à-dire la valeur maximale possible d'un pixel (si les pixels sont codés sur 8 bits, alors $d = 255$) et EQM est l'erreur quadratique moyenne définie par

$$EQM = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left(I_0(i,j) - I_r(i,j) \right)^2 .$$

La fonction MATLAB permettant de la calculer s'écrit

```
peaksnr = psnr(y,ref)
```

où `ref` est l'image de référence.

Dans le but d'avoir une bonne idée numérique de ce ratio, considérons les mêmes images représentées à la figure 1.5 à la page 47. Le tableau 1.3 donne les différentes valeurs du $PSNR$ toujours en considérant l'image originale comme référence.

Images	$PSNR$
Originale	∞
Bruitée	16,9176
Transformée	4,9740
Floue	20, 0882

TABLE 1.3 – Valeurs du $PSNR$ lorsque l'image de référence est l'image originale.

Evidemment, lorsque la deuxième image considérée est celle qui a été modifiée par une transformation géométrique, la valeur est faible par rapport aux autres : les images ne coïncident pas du tout. Nous pouvons également voir que l'image floue possède la valeur la plus élevée des trois (sans considérer le cas où la deuxième image est également l'image de référence). Cela est probablement dû au fait qu'elle ne contient pas de bruit supplémentaire comme c'est le cas pour l'image bruitée. Nous voyons donc bien que le but dans l'interprétation du $PSNR$ est qu'il soit le plus élevé possible ; qu'il tende vers l'infini.

Le chapitre suivant consiste à décrire les différentes méthodes d'optimisation applicables à notre problème. En effet, nous souhaitons optimiser une mesure de similarité entre deux images lors de nos applications concrètes.

Chapitre 2

Méthodes d'optimisation

Dans ce chapitre, nous allons nous intéresser à différentes méthodes d'optimisation qui seront étudiées dans la partie se rapportant aux applications à l'astrophotographie. En effet, celles qui sont abordées présentent toutes l'avantage de ne pas utiliser le calcul de dérivée. Il s'agit de méthodes de recherches directes (algorithmes de Nelder-Mead et Pattern Search) ou de méthodes évolutionnistes (algorithme génétique). En effet, lorsque nous devrons recaler des images, nous devrons maximiser la mesure de similarité. Or cette dernière dépend des images considérées et étant donné que la mesure va comparer les intensités des pixels, il s'agira d'une fonction discrète (presque continue puisque nous avons un grand nombre de pixels). Par conséquent, le résultat de cette mesure ne sera pas forcément lisse et donc le calcul de dérivées deviendra difficile et laborieux. Ces algorithmes étudiés peuvent donc résoudre des problèmes d'optimisation non différentiables.

Ce chapitre se divise en trois parties contenant chacune un algorithme d'optimisation. Pour chacune d'entre elles, nous décrivons la consistance de l'algorithme accompagné d'un exemple simple (une fonction à deux variables du second degré) permettant d'éclaircir la description. Finalement, nous aborderons la syntaxe de la fonction MATLAB qui a été utilisée dans l'étude.

Il a été pensé d'ajouter la méthode d'optimisation Quasi-Newton car elle avait un avantage particulier pour les méthodes sans contraintes mais elle utilise le calcul de dérivation. Or nous venons juste de souligner le fait que nous travaillerons par la suite avec des fonctions qui ne sont pas nécessairement lisses et non différentiables. Par conséquent, nous n'en ajouterons pas davantage à ce sujet et nous nous concentrons sur les trois algorithmes évoquées aux paragraphes précédents.

2.1 Algorithme de Nelder-Mead

Cette section s'est principalement basée sur les documents Mathews [10] et Wikipédia [29] concernant la description de l'algorithme ainsi que l'utilisation d'un exemple. Concernant la partie sur la fonction MATLAB utilisée [11], nous détaillerons les différentes options possibles d'exploiter.

2.1.1 Description

Il s'agit d'une méthode d'optimisation non linéaire publiée par John Nelder et Mead en 1965. Elle consiste à utiliser un *simplexe* (généralisation du triangle d'une dimension quelconque) de $n+1$ sommets dans un espace à n dimensions.

Dans un premier temps, nous définissons un simplexe quelconque. Ce dernier va subir des modifications telles qu'un agrandissement ou un rétrécissement et se déplacera dans l'espace en question et ce, jusqu'à ce que ses sommets atteignent un minimum local.

Dans le cas d'une fonction f à deux variables x et y (nous nous trouvons donc dans un espace à deux dimensions), les étapes de l'algorithme sont décrites ci-dessous.

Cas initial

- Définition d'un simplexe de départ de dimension $n+1$ (c'est-à-dire 3 dans notre cas). Les trois sommets sont $V_k = (x_k, y_k)$ où $k = 1, 2, 3$;
- Evaluation de la fonction en ces trois points : $z_k = f(x_k, y_k) = f(V_k)$ avec $k = 1, 2, 3$;
- On ordonne les différentes valeurs : $z_1 \leq z_2 \leq z_3$ et assignons une nomination pour chacun des sommets : B pour le meilleur des trois c'est-à-dire V_1 car il donne une valeur de f plus petite que les deux autres candidats ; G pour celui qui se situe entre les deux c'est-à-dire V_2 et W pour celui qui donne la plus mauvaise des valeurs entre les trois c'est-à-dire V_3.

Calcul du milieu entre les sommets B et G

$$M = \frac{B+G}{2} = \left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right).$$

Calcul d'un point de réflexion R

Il s'agit d'un point opposé à W. En effet, la valeur de la fonction va diminuer au fur et à mesure qu'on s'approche de B (ou de G). Le point R est donc l'image de W par la réflection au travers du segment $[BG]$. Pour ce faire, nous traçons le segment qui relie les points W et M, mesurons la distance d qui les sépare et la reportons au-delà du segment $[BG]$. Le point se trouvant à cette distance d de M est noté R. La figure 2.1 montre géométriquement les différentes manipulations effectuée pour calculer le nouveau point R et son expression est donnée par

$$R = M + (M - W) = 2M - W.$$

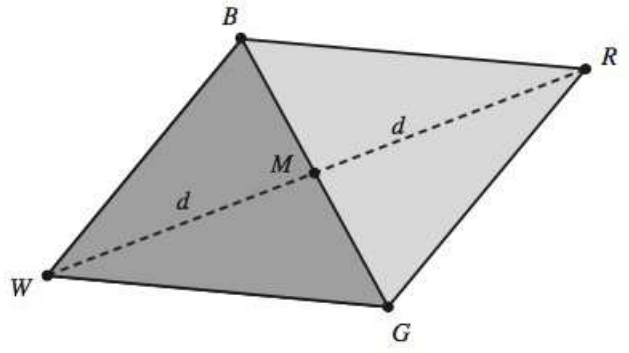


FIGURE 2.1 – Représentation du simplexe lorsque nous calculons le point de réflexion R [10].

Calcul du point d'expansion E

Dans le cas où la valeur de la fonction évaluée au point R est plus petite que celle évaluée en W, cela signifie que nous sommes dans la bonne direction ; probablement que le minimum recherché est plus loin que R ? Pour répondre à la question, nous prolongeons le segment utilisé pour construire R d'une distance d à nouveau. Cela définira le nouveau point d'extension E représenté à la figure 2.2 et celui-ci est déterminé par :

$$E = R + (R - M) = 2R - M.$$

Calcul du point de contraction C

Dans le cas où les valeurs de la fonction évaluées en R et W sont identiques, il faut tester un autre point en gardant la forme d'un triangle. Considérons alors

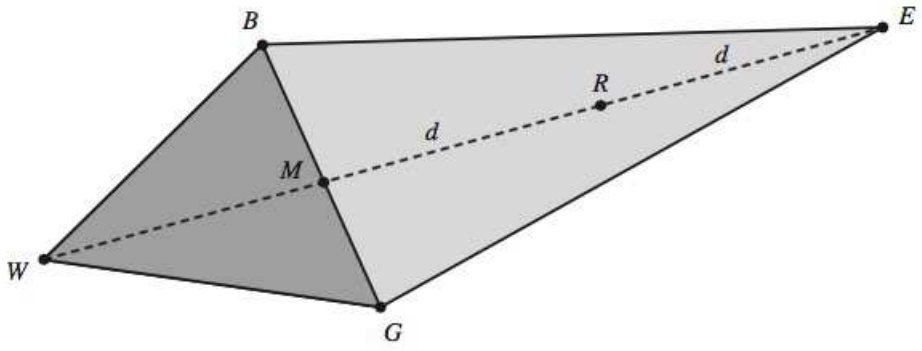


FIGURE 2.2 – Représentation du simplexe lorsque nous calculons le point d'extension E [10].

les points C_1 et C_2 étant le milieu des segments $[WM]$ et $[MR]$ respectivement (représentés à la figure 2.3). Nous garderons alors le point donnant la plus petite valeur entre les deux et le noterons C.

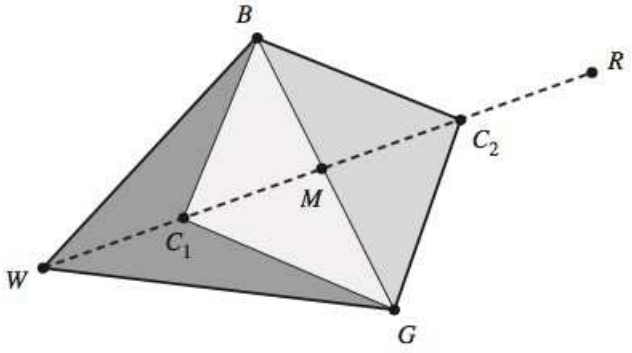


FIGURE 2.3 – Représentation du simplexe lorsque nous calculons le point de contraction C (en particulier C_1 et C_2) [10].

Cependant, la valeur évaluée au point C n'est pas forcément plus petite que celle évaluée en W. Dans ce cas-là, nous devons *amener* les points W et G vers B. Pour ce faire, nous remplaçons G par M et W par S qui est le point se situant à égale distance de W et B. Cette dernière manipulation est représentée à la figure 2.4.

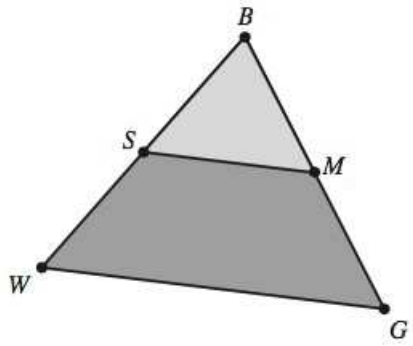


FIGURE 2.4 – Représentation du simplexe lorsque le point de contraction C ne donne pas une meilleure valeur que W [10].

2.1.2 Résumé de l'algorithme

1. Définition d'un simplexe de départ (V_1, V_2, V_3)

2. Evaluation de la fonction en ces points et triage par ordre croissant tel que
$$f(B) \leq f(G) \leq f(W) \ ;$$

3. Calcul de $M = \frac{B+G}{2}$;

4. Calcul de $R = 2M - W$;

5. L'extension du simplexe est donné dans l'algorithme 1 ;

```
if f(R) < f(W) then
    Calcul de E = 2R − M et de f(E)
    if f(E) < f(R) then
        W ← E
    else
        W ← R
    end
end
```

Algorithme 1 : Extension du simplexe.

6. La contraction du simplexe est quant à lui, donné dans l'algorithme 2 à la page 64.

if $f(W) \leq f(R)$ **then**
  Calcul de $C = \arg\min\{f(C_1), f(C_2)\}$ où $C_1 = \frac{W+M}{2}$ et $C_2 = \frac{M+R}{2}$ et de $f(C)$
  if $f(C) < f(W)$ **then**
    $W \leftarrow C$
  else
    Calcul de $V = \frac{W+B}{2}$ et $W \leftarrow V$, $G \leftarrow M$
  end
end

Algorithme 2 : Contraction du simplexe.

2.1.3 Complexité

Selon Singer [20], la complexité d'une itération de cet algorithme est linéaire $\mathcal{O}(n)$ où n est la dimension de l'espace. En effet, il est difficile d'estimer le temps de complexité d'un tel algorithme de sorte à considérer toutes les itérations. Celles-ci peuvent se stopper pour des critères d'arrêts différents en fonction du problème posé.

2.1.4 Exemple

Pour illustrer cet algorithme, considérons la fonction

$$f(x) = 3x_1^2 + x_2^2 - 12x_1 - 8x_2. \tag{2.1}$$

Remarque 2.1.1. *Cet exemple a été inspiré de Chinneck [4] et sera utilisé dans chaque algorithme présenté dans la suite de ce mémoire. En effet, cette référence était à la base une inspiration pour l'illustration de l'algorithme « pattern search » à la section 2.3.5 qui est devenu un exemple original pour tous les autres algorithmes.*

La figure 2.5 donne une idée de l'allure de la fonction ainsi que le lieu du minimum. De plus, au vu de l'expression de la fonction à minimiser, nous nous situons dans un espace à deux dimensions (deux variables).

Etape d'initialisation

Dans un premier temps, nous devons définir un simplexe initial à trois dimensions. Par exemple, prenons

$$V_1 = (0; 0), \quad V_2 = (2; 0), \quad V_3 = (0; 2).$$

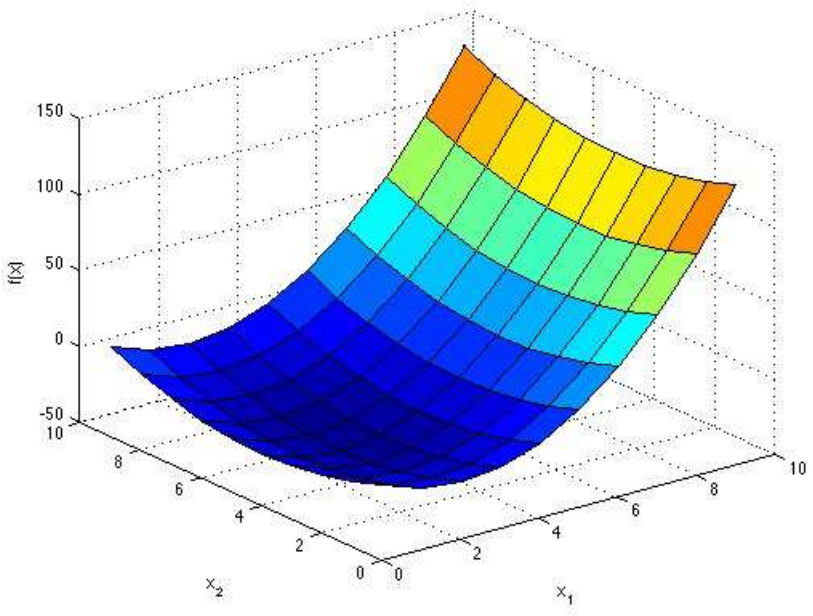


FIGURE 2.5 – Représentation graphique de la fonction $f(x) = 3x_1^2 + x_2^2 - 12x_1 - 8x_2$.

Par conséquent, la fonction évaluée en ces points donne

$$f(V_1) = 0, \quad f(V_2) = -12, \quad f(V_3) = -12.$$

La figure 2.6 montre le simplexe de départ dans l'espace à deux dimensions et contenant donc trois sommets.

Nous trions alors par ordre croissant

$$f(V_3) \leq f(V_2) \leq f(V_1).$$

Finalement pour assigner les notations

$$B = V_3, \quad G = V_2, \quad W = V_1.$$

Suite des étapes

- $k = 1$:
  - Calcul du milieu : $M = \frac{B+G}{2} = (1; 1)$;
  - Calcul de R : $R = 2M - W = (2; 2)$, d'où $f(R) = -24$;
  - Extension car $f(R) < f(W)$. Nous calculons donc $E = 2R - M = (3; 3)$ et $f(E) = -24 = f(R)$ d'où on remplace W par R.

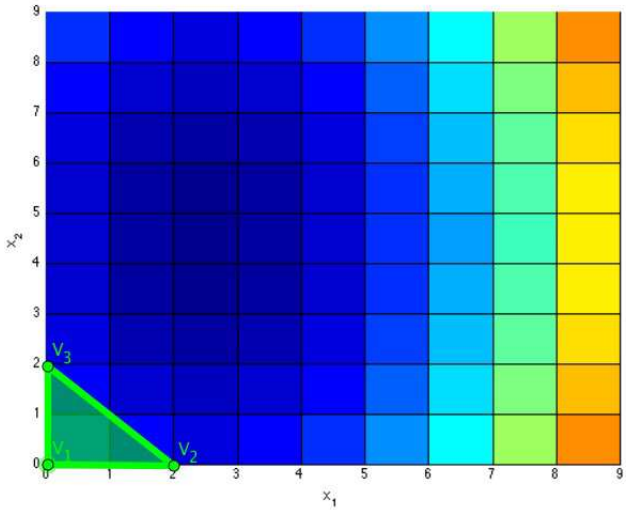


FIGURE 2.6 – Premier simplexe construit dans l'espace.

Nous obtenons donc les sommets (voir figure 2.7)

$$V_1 = (2; 2), \quad V_2 = (2; 0) \quad V_3 = (0; 2).$$

En triant les valeurs de la fonction prises en ces sommets par ordre croissant, c'est-à-dire

$$f(V_1) = -24, \quad f(V_2) = -12, \quad f(V_3) = -12,$$

nous avons

$$f(V_1) \leq f(V_2) \leq f(V_3).$$

Par conséquent, nous assignons les nouvelles notations

$$B = V_1, \quad G = V_2, \quad W = V_3.$$

- $k = 2$:
  - Calcul du milieu : $M = \frac{B+G}{2} = (2; 1)$;
  - Calcul de R : $R = 2M - W = (4; 0)$, d'où $f(R) = 0$;
  - Contraction car $f(W) < f(R)$. Nous calculons $C = \arg\min\{f(C_1), f(C_2)\}$. Or $C_1 = \frac{W+M}{2} = (2; 0, 5)$ et $C_2 = \frac{M+R}{2} = (3; 0, 5)$ D'où $f(C_1) = -15, 75$ et $f(C_2) = -12, 75$. Par conséquent, $C = C_1$. Comme $f(C) < f(W)$, nous remplaçons W par C.

Les sommets résultants (voir figure 2.8) sont donc

$$V_1 = (2; 2), \quad V_2 = (2; 0) \quad V_3 = (2; 0, 5).$$

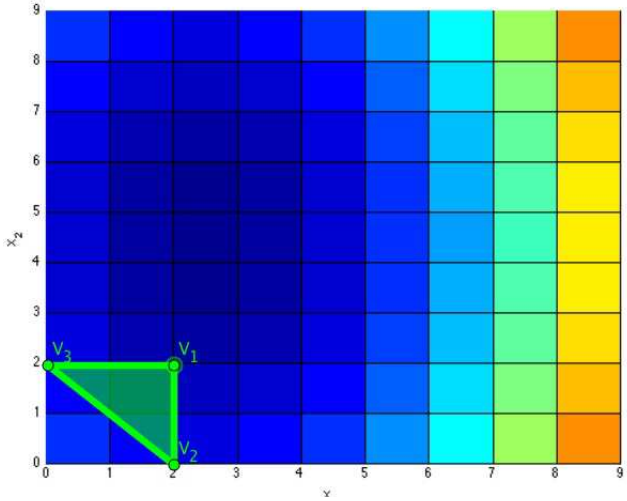


FIGURE 2.7 – Deuxième simplexe construit.

Lorsque nous trions les valeurs données en évaluant f en ces points, nous obtenons

$$f(V_1) = -24, \quad f(V_2) = -12, \quad f(V_3) = -15,75,$$

d'où

$$f(V_1) \leq f(V_3) \leq f(V_2).$$

En utilisant les nouvelles notations,

$$B = V_1, \quad G = V_3, \quad W = V_2.$$

Nous continuons de la même manière jusqu'à atteindre un certain critère, par exemple lorsque la différence de pas devient très petite. Le tableau 2.1 reprend les différents résultats de l'algorithme si nous poursuivons les itérations.

On peut remarquer que si nous continuons, nous allons nous rapprocher du minimum, c'est-à-dire $(2;4)$ où $f(2;4) = -28$.

Remarque 2.1.2. *L'exemple présenté (et ce, dans les sections suivantes également) permet de comprendre l'utilisation de l'algorithme par un exemple simple. Or nous devrons les utiliser pour des fonctions qui manipulent des images. En soi, les variables sont les paramètres de recalage (rotation et translations) et les valeurs des images ne changent pas sauf pour l'image de déformation qui est adaptée en fonction de l'évaluation de la fonction en les paramètres trouvés.*

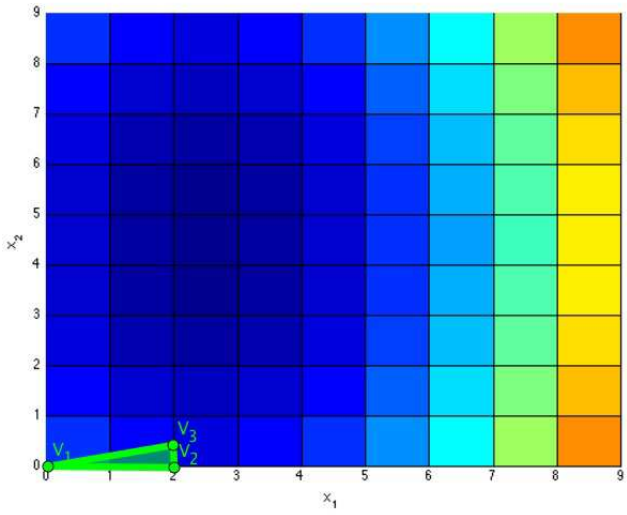


FIGURE 2.8 – Troisième simplexe construit.

2.1.5 Fonction MATLAB

Cette sous-section a été inspirée de la source [11]. La fonction MATLAB possède la syntaxe suivante

```
[x,fval,exitflag,output] = fminsearch(fun,x0,options)
```

Inputs

- `fun` est la fonction à minimiser ;
- `x0` est le point initial ;
- `options` permet de spécifier certaines précisions quant à l'optimisation cela peut être :
  - `Display` montre certains caractères du processus tels que
    - `notify` montre les résultats seulement si la fonction ne converge pas (paramètre par défaut) ;
    - `final` montre uniquement le résultat final ;
    - `off` ou `none` ne montre pas de résultat ;
    - `iter` montre les résultats à chaque itération.
  - `FunValCheck` vérifie si les valeurs de la fonction objectif sont valides :
    - `on` donne une erreur lorsque les valeurs de la fonction objectif retourne des valeurs complexes ou `NaN` ;

Itér.	B	$f(B)$	W	$f(W)$	M	R	$f(R)$	Déform.
1	$(0;2)$	-12	$(0,0)$	0	$(1;1)$	$(2;2)$	-24	Ext.
2	$(2;2)$	-24	$(0;2)$	-12	$(2;1)$	$(4;0)$	0	Contr.
3	$(2;2)$	-24	$(2,0)$	-12	$(2;1,25)$	$(2;2,5)$	$-25,75$	Ext.
4	$(2;3,75)$	$-27,94$	$(2;2)$	-24	$(2;4,5)$	$(2;7)$	-19	Contr.
5	$(2;3,75)$	$-27,94$	$(2;5,25)$	$-26,44$	$(2;3,5)$	$(2;1,75)$	$-22,93$	Contr.
6	$(2;3,75)$	$-27,94$	$(2;3,25)$	$-27,44$	$(2;4,06)$	$(2;4,88)$	$-27,23$	Contr.
7	$(2;3,75)$	$-27,94$	$(2;4,38)$	$-27,86$	$(2;3,7)$	$(2;3,03)$	$-27,0614$	Contr.
8	$(2;4,04)$	$-27,99$	$(2;3,66)$	$-27,88$	$(2;3,89)$	$(2;4,13)$	$-27,98$	Ext.

TABLE 2.1 – Résultats de l'algorithme Nelder-Mead (où « Itér. », « Déform. », « Ext. » et « Contr. » abrègent respectivement les mots « Itération », « Déformation », « Extension » et « Contraction ».

  - `off` ne montre pas d'erreur (paramètre par défaut).

- `MaxFunEvals` précise le nombre maximum d'évaluations de la fonction. La valeur définie par défaut égale `200*numberOfVariables` ;
- `MaxIter` donne le nombre maximum d'itérations lors du processus. De la même manière que `MaxFunEvals`, la valeur définie par défaut est `200*numberOfVariables` ;
- `OutputFcn` donne une ou plusieurs fonctions déterminées par l'utilisateur dont l'algorithme pour faire appel à chaque itération. Le paramètre par défaut est
  `[ ]` ;
- `PlotFcns` permet de donner le graphe de la progression de l'algorithme :
  - `@optimplotx` montre tous les points courants ;
  - `@optimplotfunccount` affiche tous les nombres d'évaluation de la fonction ;
  - `@optimplotfval` montre toutes les valeurs de la fonction ;
  - `[ ]` est le paramètre par défaut : ne montre rien.
- `TolFun` détermine un critère de terminaison de l'algorithme qui repose sur la valeur de la fonction tolérée. Par défaut, elle est définie à `1e-4` ;
- `TolX` est défini de la même manière que `TolFun` : elle détermine un second critère de terminaison qui est la valeur tolérée en `x`. La valeur par défaut est également fixée à `1e-4`. Cet algorithme s'arrête lorsque `TolFun` et `TolX` sont atteints.

Outputs

- `x` est la solution du problème d'optimisation ;
- `fval` est la valeur de la fonction évaluée en la solution calculée ;
- `exitflag` est la raison pour laquelle l'algorithme s'est arrêté :
  - Si `exitflag = 1`, alors la fonction a bien convergé vers la solution ;
  - Si `exitflag = 0`, alors le nombre maximum d'itérations fixé par `MaxIter` ou bien le nombre maximum fixé par `MaxFunEvals` ont été dépassés ;
  - Si `exitflag = -1`, alors l'algorithme s'est arrêté par la fonction résultante.
- `output` fournit des informations sur le processus qui s'est déroulé comme :
  - `iterations` donne le nombre d'itérations ;
  - `funcCount` donne le nombre d'évaluations de la fonction ;
  - `algorithm` précise l'algorithme utilisé, c'est-à-dire `Nelder-Mead simplexe direct search` ;
  - `message` fournit le message de sortie (la raison pour laquelle l'algorithme arrête d'itérer).

2.2 Algorithme génétique

Ce type d'algorithme appartient aux algorithmes évolutionnistes (Wikipédia [26]). Leur but est de trouver une solution approchée à un problème d'optimisation pour un grand nombre de fonctions et ce, en un temps raisonnable. L'algorithme est utilisable pour des fonctions discrètes ou non différentiables. L'idée centrale de ce type d'algorithme est la notion de *sélection naturelle* et l'applique à une population de solutions potentielles au problème donné. En effet, cette population de solutions sera la mieux adaptée au problème comme les individus au fil du temps se sont adaptés à leur environnement et aux conditions de vie où ils étaient confrontés.

Les auteurs Vallée et Yilildizoglu [24], et Bodenhofer [3] ont inspiré l'ensemble de la section ainsi que le mémoire de D. Nicolay [18].

2.2.1 Analogie avec la biologie

Etant donné que l'algorithme s'inspire du mécanisme de l'évolution, théorie issue de la biologie, il semble naturel de spécifier le vocabulaire employé :

- Un *gène* est un morceau de l'ADN et correspond à une information génétique particulière. Ils sont responsables de l'hérédité, qui contrôlent un caractère ou des aptitudes propres à un organisme (nombre de membres, couleur des yeux, de la peau, ...). Un exemple parmi d'autres est le gène qui détermine le groupe sanguin (système ABO) auquel on appartient ;
- Un *allèle* est une version variable d'un même gène. En utilisant le même exemple que dans le point précédent, le gène du groupe sanguin, nous avons les allèles codominants A et B, et l'allèle récessif O. Par conséquent, un individu possédant les allèles A et B appartiendra au groupe AB tandis qu'un individu appartiendra au groupe O que s'il possède les deux allèles O ;
- Un *chromosome* est une structure composée d'ADN. Le corps humain en possède 23 paires et chacun possède des formes différentes. Ils sont porteurs des gènes ;
- Le *génotype* est l'ensemble des caractéristiques génétiques d'un individu et est unique pour tous (sauf cas de vrais jumeaux). Il s'agit de la composition des allèles de tous les gènes ;
- Le *phénotype* est l'ensemble des caractères observables d'un individu ; il correspond à la réalisation du génotype (expression des gènes) et aux effets du milieu environnant ;
- Un *individu* est composé d'un ou plusieurs chromosomes ;
- Une *population* est composée d'individus ;
- La *fitness* est une mesure quantitative du niveau d'adaptation d'un individu dans son milieu ; cela va permettre d'appliquer la sélection naturelle : on cherche à la maximiser.

2.2.2 Principes

Dans un premier temps, nous créons une population de départ de manière aléatoire afin d'avoir une bonne dispersion. Ensuite, l'algorithme se base sur les trois principes suivants :

1. Sélection ;
2. Enjambement ;
3. Mutations.

Ces étapes sont itérées jusqu'à ce qu'un critère d'arrêt soit atteint.

2.2.3 Sélection

Comme dit précédemment, l'algorithme se base sur la sélection naturelle. En effet, les individus les mieux adaptés gagnent la compétition et se reproduisent ;

tandis que les moins adaptés meurent avant la reproduction. Dans ce cas-ci, étant donné que la sélection se fait par l'homme, la sélection sera artificielle et non pas naturelle comme annoncé au départ ; elle se fera grâce à la fitness correspondante à la fonction objectif f.Il existe plusieurs méthodes de sélection qui sont détaillées ci-dessous.

Roulette

Cette méthode consiste à associer à chaque individu un sous-intervalle de $[0, 1]$ de longueur proportionnelle à sa fitness. Par conséquent, les individus les mieux adaptés auront une plus grande longueur d'intervalle.

Il s'agit de la sélection par roulette car chaque section d'un disque pourrait représenter un individu et son étendue reflèterait sa proportionnalité avec la fitness. De plus, cette dernière est associée à la probabilité de sélection : soit f_i la fitness de l'individu i alors la probabilité que cet individu soit sélectionné est donnée par

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i},$$

où N est le nombre total d'individus au sein de la population. De cette manière, lorsqu'on sélectionnera au hasard un individu représenté par une section de la roue, on aura plus de chance d'en sélectionner un avec une meilleure adaptation à son environnement.

Cependant, cette méthode comporte l'inconvénient d'être fort sensible à la taille de population ; on n'aura pas forcément un bon tirage dans le cas de petites populations. De plus, la variance de la fonction fitness a un impact sur la roulette. En effet, lorsque la variance est trop petite, on revient à un tirage au sort proprement dit. Tandis que si la variance est trop grande, nous n'allons sélectionner que les meilleurs, ne plus avoir une certaine diversité au sein de la population et par conséquent, passer à côté des individus les plus intéressants.

Une manière d'enlever ces inconvénients est le scaling (réduit ou augmente l'écart de la fitness et permet de diminuer l'écart entre les individus) et le sharing (pénalise les fitness communes à un grand nombre d'individus).

Tournoi

Cette deuxième stratégie consiste à prendre deux individus dans une population au hasard et leur attribuer une probabilité p d'être choisi à celui qui a la plus grande fitness et une probabilité $1 - p$ à son adversaire où $p \in]\frac{1}{2}, 1]$, fixé. Ensuite, on itère le processus jusqu'à ce que la nouvelle population soit complète.

2.2.4 Enjambement

Il s'agit du processus lorsque deux chromosomes s'échangent une (ou des) portion(s) d'ADN et s'appelle également « cross-over ». Sa probabilité d'apparition est un paramètre de l'algorithme génétique et est donc comprise entre 0 et 1 (strictement). Néanmoins, ces enjambements peuvent être multiples : dans ce cas là, il y a plusieurs points d'enjambement. Cette étape correspond donc à la reproduction ; il s'agit bien du moment où les séquences d'ADN vont se croiser via ces enjambements. Le résultat serait donc d'obtenir des enfants au moins aussi bons (voire meilleurs) que leurs parents.

2.2.5 Mutations

Un gène peut substituer à un autre de manière aléatoire. Il faut donc définir un taux de mutation (généralement compris entre 0.001 et 0.1). La mutation sert à ce que l'algorithme ne tombe pas dans une convergence prématurée ; c'est pour cela que le taux de mutation ne doit pas être élevé. Le but de cette étape est donc de modifier le code génétique d'un individu en ajoutant une information qui n'était pas présente via l'ADN des parents. Cet ajout d'information génétique amènerait une population encore meilleure que celle engendrée par les parents de départ.

De plus, cette étape se fait sur la population de départ pour ne pas abîmer les individus créés à partir de l'étape d'enjambement. Dans un premier temps, nous fixons la probabilité de mutation à p_m ; chaque gène a une probabilité p_m d'être muté. Par conséquent, pour chaque bit, nous tirons un nombre compris entre 0 et 1 et si ce chiffre est inférieur à p_m, alors nous effectuons la mutation en ce gène. Il existe plusieurs méthodes de mutations : soit un seul gène est muté (1-inversion), soit plusieurs gènes (n-inversions), soit tous (inversion totale).

2.2.6 Complexité

La complexité de cet algorithme dépend des méthodes utilisées (méthodes de sélection, de mutation, ...) et en plus de cela, elle dépend de la fonction fitness. Selon F. Lobo et al. [8],typiquement, le nombre d'évaluations de la fonction fitness dépend du produit du nombre de générations avec la taille de la population, pour une taille de population fixée. Cependant cette dernière peut être modifiée au cours du processus soit de manière linéaire, soit de manière exponentielle.

Numéro	Chromosome x	Chromosome y	(x, y)	$f(x, y)$
1	00011	00011	$(3, 3)$	-24
2	00001	00010	$(1, 2)$	-21
3	00100	00101	$(4, 5)$	-15
4	01000	00110	$(8, 6)$	84
5	00111	01010	$(7, 10)$	83

TABLE 2.2 – Première génération des individus.

2.2.7 Exemple

Considérons la fonction utilisée dans le cas de l'algorithme de Nelder-Mead dans la section précédente 2.1.4 (à l'équation (2.1)) où la représentation graphique est référée à la figure 2.5.

Création de la population

Dans un premier temps, nous devons avoir une population initiale créée aléatoirement.Remarquons qu'il s'agit d'une fonction dans un espace à deux dimensions. Par conséquent, les individus engendrés seront des couplets de type (x, y) où x est la valeur au niveau de l'axe des abscisses et y est la valeur au niveau de l'axe des ordonnées. Par exemple, considérons la population à 5 individus codés sur 5 bits dans le tableau 2.2.

Sélection

Pour illustrer l'algorithme, prenons la sélection par roulette de sorte à assigner à chaque individu une probabilité d'être tiré au hasard par la roue. Plus sa probabilité est élevée, plus il a de chance d'être sélectionné et donc de générer des enfants ; il s'agit des individus qui s'adaptent le mieux. Les informations sont données dans le tableau 2.3. La figure 2.9 donne une idée de la représentation de la roulette biaisée (les pourcentages ont été arrondis à l'unité). Nous pouvons donc constater que les individus 1 et 2 (et 3) ont plus de chance d'être sélectionnés par rapport aux individus 4 et 5. De fait, ces derniers donnent une valeur de la fonction objectif beaucoup plus élevée que les trois autres ; pour rappel, nous cherchons à minimiser la fonction.

Par conséquent, si nous tirons au hasard 5 fois, nous obtenons l'exemple à la table 2.4.

Numéro	Fitness	Pourcentage
1	224	25, 08%
2	221	24, 75%
3	215	24, 08%
4	116	12, 99%
5	117	13, 10%
Somme	893	100%

TABLE 2.3 – Aperçu de la fitness pour chacun des individus de la population.

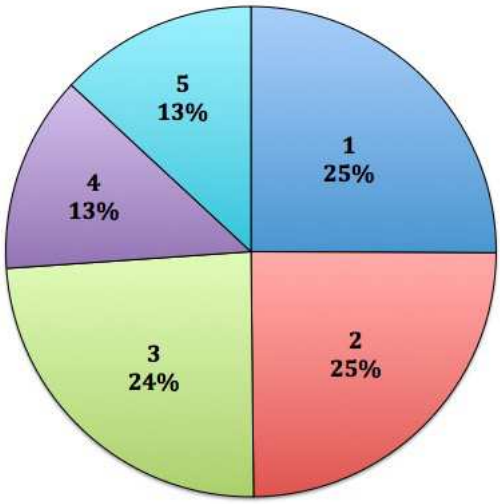


FIGURE 2.9 – Sélection par le procédé de la roulette.

Enjambement

Supposons que les chromosomes 1 et 2 ont été choisis pour la reproduction et déterminons le point d'enjambement au niveau du 4^e bit. Les croisements se font au niveau des abscisses et des ordonnées respectivement. Un exemple est donné dans le tableau 2.5.

Par conséquent, les enfants générés par ce croisement sont $(3, 2)$ et $(1, 3)$ et les valeurs de la fonction objectif sont -21 et -24 respectivement. Nous restons donc dans de bonnes valeurs de la fonction objectif (nous n'augmentons pas par rapport aux parents) au vu des valeurs de la premières génération.

Numéro	Chromosome x	Chromosome y	(x, y)	$f(x, y)$
1	01000	00110	$(8, 6)$	84
2	00011	00011	$(3, 3)$	-24
3	00011	00011	$(3, 3)$	-24
4	00100	00101	$(4, 5)$	-15
5	00001	00010	$(1, 2)$	-21

TABLE 2.4 – Résultat de la sélection des individus.

	Numéro	Parents					Enfants					Valeurs
x	1	0	0	0	1	1	0	0	0	1	1	3
	2	0	0	0	0	1	0	0	0	0	1	1
y	1	0	0	0	1	1	0	0	0	1	0	2
	2	0	0	0	1	0	0	0	0	1	1	3

TABLE 2.5 – Croisement des gènes des parents.

Mutation

Pour cette étape, prenons l'exemple de prendre le chromosome 1 et mutons le deuxième bit dans le cas des coordonnées en abscisses et le troisième bit dans le cas des coordonnées en ordonnées. Nous obtenons alors le tableau 2.6 où la nouvelle valeur de la fonction fitness devient 212 qui est bien meilleure que la valeur précédente sans la mutation.

Itération des étapes

Après avoir généré une nouvelle population grâce à l'enjambement et à la mutation de chaque individu, nous continuons le même processus que celui décrit précédemment jusqu'à atteindre un certain nombre d'itérations ou bien un autre critère d'arrêt possible.

	Enfants					Après mutation					Valeurs	$f(x, y)$
x	0	1	0	0	0	0	0	0	0	0	0	-12
y	0	0	1	1	0	0	0	0	1	0	2	

TABLE 2.6 – Mutation des gènes de l'enfant n°1.

2.2.8 Fonction MATLAB

La syntaxe de la fonction d'optimisation par l'algorithme génétique sur MATLAB [12] est décrite de la manière suivante :

```
[x,fval,exitflag,output,population,scores] =
ga(fitnessfcn,nvars,A,b, Aeq,beq,LB,UB,nonlcon,IntCon,options)
```

Inputs

- `fitnessfcn` est la fonction fitness ;
- `nvars` est un entier positif contenant le nombre de variables du problème à optimiser ;
- `A` et `b` représentent les contraintes d'inégalités linéaires de la forme `A*x <= b` où `A` encode les m inégalités linéaires, `b` est un vecteur de taille m et `x` est un vecteur de taille n (nombre de variables) ;
- `Aeq` et `beq` possèdent les même rôles que `A` et `b` respectivement sauf qu'ils interviennent dans le cas des contraintes d'égalités de la forme `Aeq*x = beq` (les dimensions sont identiques) ;
- `LB` donne les bords inférieurs tels que, pour tout `i`, `x(i) >= LB` ;
- `UB` donne les bords supérieurs tels que, pour tout `i`, `x(i) <= UB` ;
- `nonlcon` est une fonction qui accepte comme argument `x` et retourne deux tableaux : `c` et `ceq` qui représentent les contraintes d'inégalités et d'égalités non linéaires respectivement en `x` tels que `c(x) <= 0` et `ceq(x) = 0` ;
- `options` :
  - `ConstraintTolerance` donne la faisabilité en tenant compte des contraintes non linéaires et `max(sqrt(eps),ConstraintTolerance)` détermine la faisabilité dans le cas des contraintes linéaires. La valeur par défaut est 10^{-3} ;
  - `CreationFcn` est une fonction qui crée une population initiale. Cela peut se faire suivant différentes méthodes :
    - `@gacreationuniform` est la fonction par défaut ;
    - `@gacreationlinearfeasible` est la fonction par défaut dans le cas des contraintes entières.
  - `CrossoverFcn` est la fonction qui va permettre à l'algorithme de générer une nouvelle population. Il existe également plusieurs méthodes :
    - `@crossoverscattered` ;

  - `@crossoverheuristic` ;
  - `@crossoversinglepoint` ;
  - `@crossovertwopoint` ;
  - `@crossoverarithmetic`.

- `CrossoverFraction` est la fraction de la population à la génération suivante sans inclure les enfants élites qui eux sont créés par l'étape d'enjambement. La valeur par défaut est $0,8$;
- `Display` montre certains caractères du processus tels que
  - `off` ou `none` ne montre pas de résultat ;
  - `iter` montre les résultats à chaque itération ;
  - `diagnose` ;
  - `final` montre uniquement le résultat final (valeur par défaut).
- `EliteCount` est un entier positif qui spécifie le nombre d'individus qui vont survivre à la prochaine génération. La valeur par défaut est `ceil(0.05*PopulationSize)` (où la fonction `ceil()` arrondit l'argument à l'entier supérieur) ;
- `FitnessLimit` si la fonction fitness atteint cette valeur, l'algorithme s'arrête. Par défaut, la valeur est `Inf` ;
- `FitnessScalingFcn` manipule une fonction qui met à l'échelle les valeurs de la fonction fitness. Il en existe plusieurs :
  - `@fitscalingrank` est la fonction par défaut ;
  - `@fitscalingshiftlinear` ;
  - `@fitscalingprop` ;
  - `@fitscalingtop`.
- `FunctionTolerance` permet d'arrêter l'algorithme lorsque le changement dans la moyenne relative des valeurs de la fonction fitness sur les `MaxStallGenerations` générations est plus petit ou égal à `FunctionTolerance`. Dans le cas où `StallTest` est de type `geometricWeighted`, alors le processus s'arrête lorsque le changement de la moyenne relative pondérée est plus petite ou égale à `FunctionTolerance`. La valeur par défaut est 10^{-6} ;
- `HybridFcn` manipule une fonction qui continue l'optimisation lorsque l'algorithme génétique a terminé. Cela peut être les fonctions suivantes :
  - `@fminsearch` ;
  - `@patternsearch` ;
  - `@fminunc` ;
  - `@fmincon` ;

  - Par défaut, il n'y a pas de fonction ajoutée.

- `InitialPopulationMatrix` contient une matrice avec la population initiale pour démarrer l'algorithme. Le nombre de lignes correspond à la taille de la population (`PopulationSize`) et le nombre de colonnes correspond au nombre de variables. Il est également possible d'y assigner une population initiale possédant un nombre de lignes plus petit que `PopulationSize`. Si c'est le cas, l'algorithme fait appel à `CreationFcn` pour générer les membres de la population manquants. Par défaut, il n'y a pas de matrice définie ;
- `InitialPopulationRange` est une matrice ou un vecteur spécifiant l'intervalle des individus dans la population initiale. Par défaut, il s'agit de $[-10; 10]$ dans le cas de composantes non bornées ; $[-10^4 + 1; 10^4 + 1]$ pour les problèmes avec contraintes entières et dont les composantes sont non bornées ; et $[lb; ub]$ pour les composantes bornées ;
- `InitialScoresMatrix` représente les scores initiaux pour déterminer la fonction fitness. Il s'agit d'une matrice ou d'un vecteur dont le nombre de lignes correspond à la taille de la population et le nombre de colonnes correspond au nombre de fonctions fitness (dans notre cas, il s'agit d'une seule fonction car nous ne faisons pas de l'algorithme multi-objectif). De la même manière que pour `InitialPopulationMatrix`, nous pouvons assigner une matrice de taille plus petite que `PopulationSize` mais alors, l'algorithme va remplir les scores lorsqu'il évaluera la fonction fitness. Par défaut, la matrice (ou le vecteur) est vide ;
- `MaxGenerations` est le nombre maximum de générations. La valeur par défaut correspond à `100*numberOfVariables` ;
- `MaxStallGenerations` est la valeur pour laquelle l'algorithme stoppe si la variation relative moyenne de la meilleure valeur de la fonction fitness sur les `MaxStallGeneration` générations est plus petite ou égale à `FunctionTolerance`. Si `StallTest` est de type `geometricWeighted`, alors l'algorithme s'arrête lorsque la variation relative moyenne pondérée est plus petite ou égale à `FunctionTolerance`. Par défaut, la valeur égale 50 ;
- `MaxStallTime` permet de stopper l'algorithme lorsqu'il ne trouve pas d'amélioration dans la fonction objectif pendant `MaxStallTime` secondes (mesurées grâce aux fonctions `tic` et `toc`). La valeur par défaut est `Inf` ;
- `MaxTime` permet de stopper l'algorithme lorsque le temps `MaxTime` est dépassé. De plus, l'algorithme peut franchir cette limite lorsqu'une itération prend un temps considérable à être réalisée (par défaut, la valeur est `Inf`) ;

- `MutationFcn` définit la fonction qui va engendrer les enfants. Dans le cas de l'algorithme à un simple objectif, la fonction est `@mutationgaussian` par défaut ou nous pouvons utiliser la fonction `@mutationuniform` ;
- `NonlinearConstraintAlgorithm` représente l'algorithme avec contraintes non linéaires (`auglag`)
- `OutputFcn` sont les fonctions que l'algorithme appelle à chaque itération (par défaut, il n'y a pas de fonctions définies) ;
- `PlotFcn` permet de donner le graphe de la progression de l'algorithme :
  - `@gaplotdistance` ;
  - `@gaplotgenealogy` ;
  - `@gaplotselection` ;
  - `@gaplotscorediversity` ;
  - `@gaplotscores` ;
  - `@gaplotstopping` ;
  - `@gaplotmaxconstr` ;
  - `@gaplotbestf` ;
  - `@gaplotbestindiv` ;
  - `@gaplotexpectation` ;
  - `[ ]` est le paramètre par défaut : ne montre rien.
- `PopulationSize` détermine la taille de la population. La valeur par défaut est 50 lorsque `numberOfVariables <= 5`, sinon c'est 200 ;
- `PopulationType` spécifie le type de population. Par défaut, la population est un vecteur de valeurs de type `double` (`doubleVector`). D'autres valeurs sont possibles comme `bitstring` ou `custom` mais dans notre cas, les contraintes de ce type seront ignorées par l'algorithme (il ne s'agit pas d'un algorithme multi-objectif) ;
- `SelectionFcn` est la fonction qui sélectionne les parents des étapes d'enjambement et de la mutation. Il existe plusieurs méthodes :
  - `@selectionstochunif` qui est la fonction utilisée par défaut ;
  - `@selectionremainder` ;
  - `@selectionuniform` ;
  - `@selectionroulette`.
- `UseParallel` calcule les fonctions fitness et les contraintes non linéaires en parallèle (par défaut, il ne le fait pas, la valeur est donc `false`) ;
- `UseVectorized` spécifie si les fonctions sont vectorisées (`false` par défaut).

Outputs

- `x` est la meilleure solution trouvée ;
- `fval` est la valeur de la fonction objectif évaluée en `x` ;
- `exitflag` est la raison pour laquelle l'algorithme s'est arrêté :
  - Si `exitflag = 1` : dans le cas où le problème d'optimisation était sans contrainte non linéaire, la moyenne des changements cumulatifs de la fonction fitness sur les `MaxStallGenerations` générations est plus petite que `FunctionTolerance` et la contrainte de violation est plus petite que `ConstraintTolerance`. Dans le cas où le problème est avec des contraintes non linéaires, cela signifie que l'amplitude de la mesure de complémentarité est plus petite que `sqrt(ConstraintTolerance)`, le sous-problème est résolu en utilisant une tolérance plus petite que `FunctionTolerance`, et la contrainte de violation est plus petite que `ConstraintTolerance` ;
  - Si `exitflag = 3` : la valeur de la fonction fitness n'a pas changé dans les `MaxStallGenerations` générations et la contrainte de violation est plus petite que `ConstraintTolerance` ;
  - Si `exitflag = 4` : la longueur du pas est plus petite que la précision machine (soit $2,2204.10^{-16}$) et la violation de contrainte est plus petite que `ConstraintTolerance` ;
  - Si `exitflag = 5` : la limite de la fonction fitness `FitnessLimit` est atteinte et la contrainte de violation est plus petite que `ConstraintTolerance` ;
  - Si `exitflag = 0` : le nombre maximum de générations `MaxGenerations` est atteint ;
  - Si `exitflag = -1` : l'optimisation s'est arrêtée par une fonction résultante ou un graphe de la fonction ;
  - Si `exitflag = -2` : il n'y a pas eu de point trouvé ;
  - Si `exitflag = -4` : la limite de temps de décrochage `MaxStallTime` est atteinte ;
  - Si `exitflag = -5` : la limite de temps `MaxTime` est atteinte.
- `output` fournit des informations sur les générations et le processus qui s'est déroulé comme :
  - `problemtype`, précise le type de problème considéré comme
    - `unconstrained` (sans contrainte) ;
    - `boundconstraints` (avec contraintes aux bords) ;
    - `linearconstraints` (avec contraintes linéaires) ;
    - `nonlinearconstr` (avec contraintes non linéaires) ;
    - `integerconstraints` (avec des contraintes entières).

  - `rngstate` donne l'état du générateur de nombres aléatoires juste avant que l'algorithme ne démarre ;
  - `generations` donne le nombre de générations calculées ;
  - `funccount` donne le nombre d'évaluations de la fonction fitness ;
  - `message` fournit le message contenant la raison pour laquelle l'algorithme s'est arrêté ;
  - `maxconstraint` donne la contrainte de violation maximale si jamais cela devait se préciser.

- `population` est une matrice dont les lignes contiennent les membres de la population finale ;
- `scores` est un vecteur colonne contenant les valeurs de la fonction fitness de la population finale dans le cas des problèmes de type `integerconstraints`.

2.3 Pattern Search

Il s'agit d'une famille de méthode d'optimisation numérique qui a l'avantage de ne pas utiliser le gradient et peut être utilisée sur des fonctions non continues ou non différentiables. Cette méthode est également connue sous le nom de méthode d'optimisation de *recherche directe* ou *boîte noire*. Il existe plusieurs méthodes *Pattern Search* dont l'une des plus connues provient de Hooke et Jeeves publiée en 1961. Cette dernière est décrite dans les sous-sections qui suivent. En effet, elle peut être décrite suivant deux grandes étapes : l'étape de recherche (exploratoire) et l'étape de sondage.

Le nom « Pattern Search » provient du fait que lorsque nous allons rechercher à améliorer les valeurs de la fonction objectif, nous allons utiliser des « formes » de points ; ce qui est indépendant de la considération de la fonction (Torczon [22]). Par exemple, dans l'exemple d'application que nous présenterons dans la sous-section 2.3.5, nous considérons la forme de points comme étant quatre vecteurs orientés dans des directions opposées.

Evidemment, le problème à résoudre d'une telle méthode d'optimisation est

$$\min_{x \in \mathbb{R}^n} f(x).$$

Cette section a été principalement inspirée par les auteurs Audet [2], Torczon[22], [23] et MATLAB [15] pour la fonction utilisée.

2.3.1 Recherche exploratoire

Il s'agit d'une recherche grossière de la direction du gradient. En effet, le but est de trouver une direction (et pas forcément la meilleure) qui minimise la valeur de la fonction objectif. Pour ce faire, on bouge le point courant d'un petit pas Δ_k dans chaque direction.

2.3.2 Sondage

Avant de passer à l'itéré suivant, nous devons *sonder* les valeurs des points avoisinants l'itéré courant. En effet, il faut qu'on sache si les autres voisins ne donnent pas une meilleure valeur plus basse de la fonction objectif.

Nous sommes alors face à deux cas de figures :

- Si aucun des points sondés ne donne une meilleure valeur de la fonction (une valeur supérieure à celle de l'itéré courant), alors nous pouvons dire que le test est un *échec* et nous devons ajuster la grille de valeurs (cela signifie que la grille n'est pas assez affinée) et assigner au nouvel itéré, la valeur de l'itéré courant c'est-à-dire $x_{k+1} = x_k$.
- Si, au contraire, l'itération est un succès (c'est-à-dire que nous avons trouvé un point tel que la valeur de la fonction objectif diminue strictement), nous gardons la longueur du paramètre Δ_k inchangée, ou bien il est possible de l'augmenter.

Après cela, le processus est réitéré. Dans ce cas, nous cherchons une grille de valeurs plus grossière grâce à la longueur de pas $\tau^{m_k^+}\Delta_k$ pour un certain $\tau^{m_k^+} \geq 1$ où $m_k^+ \geq 0$ est un entier borné supérieurement par $m_{max} \geq 0$.

2.3.3 Mise à jour de la longueur du pas

Pour pouvoir définir la grille de valeurs, nous avons besoin de la longueur du pas. En d'autres termes, c'est ce qui va nous permettre de définir la grille de valeurs via laquelle nous allons sonder les valeurs de la fonction objectif.

Dans un premier temps, considérons un ensemble fini de matrices $\mathcal{S}$ dont l'ensemble des colonnes génère l'espace $\mathbb{R}^n$.

La grille de valeurs courantes M_k est définie par

$$M_k = \{x_k + \Delta_k S z : z \in \mathbb{Z}^{n_S}, S \in \mathcal{S}\},$$

où x_k est l'itéré courant, $\Delta_k > 0$ est la longueur du pas de la grille de valeur et n_S est le nombre de colonnes de la matrice S. Cette notation permet de

déterminer les différentes valeurs appartenant à la grille ainsi que sa taille (via Δ_k). Grâce à cette notation, nous pouvons déterminer l'ensemble des valeurs qui vont être sondées, c'est-à-dire

$$\{x_k + \Delta_k s : s \text{ est une colonne de } S\}.$$

À présent que nous avons défini la grille de valeurs M_k, nous pouvons nous concentrer sur la mise à jour de la taille de la grille, c'est-à-dire Δ_k. Face aux deux cas de figures que nous avons rencontrés dans la sous-section précédente, nous modifions la longueur du pas Δ_k selon que nous avons réussi le test ou que nous l'avons échoué.

- Si le test durant la phase de sondage est un échec, alors la taille de la grille est trop vaste et nous devons l'affiner. Pour ce faire, nous assignons la valeur $\tau^{m_k^-}\Delta_k$ à Δ_{k+1} où $0 < \tau^{m_k^-} < 1$ et $\tau > 1$ reste constant tout au long du processus ; cela permet de diminuer la taille de la grille.

  De plus, $m_k^- \leq -1$ est un entier borné inférieurement par $-m_{max} \leq 0$. En effet, dans le cas où le critère d'arrêt de l'algorithme provient de la longueur de la taille de la grille, nous avons besoin de cette valeur m_{max}. Par exemple, en utilisant le logiciel MATLAB, par défaut, l'algorithme s'arrête lorsque la taille de la longueur de pas Δ_k passe en dessous de la valeur 10^{-6}. Si $\tau = 2$ alors $2^{-19} = 1.907310^{-6}$ (d'où $m_{max} = 19$) et $2^{-20} = 9.536710^{-7} < \tau^{-m_{max}}$. Si nous avons atteint ce seuil, cela signifiera que nous sommes extrêmement proches de la solution (si pas exactement).

- Si le test est une réussite, alors nous agrandissons le pas (de telle manière à accélérer le processus au cas où nous serions loin de la solution) ou bien il reste inchangé. Nous assignons donc la nouvelle valeur $\tau^{m_k^+}\Delta_k$ à Δ_{k+1} où $\tau^{m_k^+} \geq 1$ et $m_k^+ \geq 0$ est un entier borné supérieurement par $m_{max} \geq 0$. De cette manière, au fil des itérations, il existe un entier $r_k \in \mathbb{Z}$ tel que la longueur de pas courante Δ_k peut s'exprimer en fonction de la première longueur de pas Δ_0 c'est-à-dire $\Delta_k = r_k\Delta_0$. De plus l'itéré suivant peut s'exprimer comme la combinaison de toutes les directions prises à partir de la condition initiale x_0 c'est-à-dire

$$x_{k+1} = x_0 + \sum_{i=1}^{k} \Delta_i S_i z_i \qquad \text{où } z_i \in \mathbb{Z}^{n_{S_i}}.$$

2.3.4 Algorithme

1. Initialisation :
   - Considérons la condition initiale x_0 telle que $f(x_0) < \infty$, la première grille de valeurs M_0 sur $\mathbb{R}^n$ définie par la longueur de pas $\Delta_0 > 0$ (et x_0) ;
   - Etablissons le compteur d'itérations k à 0.
2. Recherche exploratoire : On détermine la grille de valeurs M_k grâce à la longueur de pas Δ_k autour de l'itéré courant x_k.
3. Sondage : Evaluation de la fonction objectif sur l'ensemble des voisins de l'itéré courant x_k (soit $f(M_k)$) et comparaison de la valeur de la fonction objectif avec $f(x_k)$.
4. Mise à jour du paramètre :
   - Si le test est *réussi*, alors nous avons un nouvel itéré x_{k+1} tel que $f(x_{k+1}) < f(x_k)$. Dans ce cas, la longueur de pas devient $\Delta_{k+1} \leq \Delta_k$ tel que $\Delta_{k+1} = \tau^{m_k^+} \Delta_k$.
   - Si le test est un *échec*, alors nous ne changeons pas la valeur de l'itéré courant c'est-à-dire $x_{k+1} = x_k$. Par conséquent, la longueur de pas devient $\Delta_{k+1} < \Delta_k$ telle que $\Delta_{k+1} = \tau^{m_k^-} \Delta_k$.
   - On passe à l'itération suivante $k \leftarrow k + 1$ et on recommence le processus à partir de l'étape 2.

2.3.5 Exemple

La fonction à minimiser est identique à celle présentée dans l'algorithme de Nelder-Mead, c'est-à-dire l'équation (2.1). De plus, nous laissons le lecteur se référer à la figure 2.5 afin de garder l'idée du lieu du minimum ainsi que l'allure de la fonction.

Pour résoudre ce problème d'optimisation en utilisant cet algorithme, utilisons les mêmes constantes que le logiciel MATLAB [15] utilise (étant donné que les applications à l'astrophotographie utiliseront ce logiciel) :

Appellation MATLAB	Constantes	Valeurs	
InitialMeshSize	Δ_0	1	
MeshContractionFactor	τ	2	$\tau^{m_k^-} = 2^{-1}$
	$\forall k, m_k^-$	-1	
MeshExpansionFactor	τ	2	$\tau^{m_k^+} = 2$
	$\forall k, m_k^+$	1	

Avant de commencer à résoudre, nous devons connaître la méthode de sondage choisie. Par défaut, MATLAB utilise *GPS Positive basis 2N* qui consiste à prendre deux fois le nombre de variables indépendantes de la fonction objectif. Dans le cas de l'exemple, on aura 4 vecteurs :

$$S_1 = \begin{bmatrix}1\\0\end{bmatrix}, \quad S_2 = \begin{bmatrix}0\\1\end{bmatrix}, \quad S_3 = \begin{bmatrix}-1\\0\end{bmatrix} \text{ et } \quad S_4 = \begin{bmatrix}0\\-1\end{bmatrix}.$$

Etape d'initialisation :

- $x_0 = (1,1)$ donc $f(x_0) = -16$;
- $\Delta_0 = 1 > 0$;
- $S = \begin{bmatrix}1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{bmatrix}$ où les colonnes génèrent $\mathbb{R}^2$;
- $$\begin{aligned} M_0 &= \{x_0 + \Delta_0 S z : z \in \mathbb{Z}^4, S \in \mathcal{S}\} \\ &= \{(1,1)+(1,0);(1,1)+(0,1);(1,1)+(-1,0);(1,1)+(0,-1)\} \\ &= \{(2,1);(1,2);(0,1);(1,0)\}. \end{aligned}$$

Etape de recherche : Calcul de $f(x_k + s_k)$

- $\underline{k=0}$:
  - $f((1,1)+(1,0)) = f(2,1) = -19$;
  - $f((1,1)+(0,1)) = f(1,2) = -21$;
  - $f((1,1)+(-1,0)) = f(0,1) = -7$;
  - $f((1,1)+(0,-1)) = f(1,0) = -9$.

  Nous avons $f(x_0) = f((1,1)) = -16 < f(1,2) = -21$. Par conséquent, nous pouvons continuer avec ce nouvel itéré $(1,2)$ c'est-à-dire $x_1 = (1,2)$ et augmenter la longueur de pas $\Delta_1 = \tau^{m_0^+}\Delta_0 = 2$ puisque le test est un succès (voir la figure 2.10).

- $\underline{k=1}$:
  - $f((1,2)+(2,0)) = f(3,2) = -21$;
  - $f((1,2)+(0,2)) = f(1,4) = -25$;
  - $f((1,2)+(-2,0)) = f(-1,2) = 3$;
  - $f((1,2)+(0,-2)) = f(1,0) = -9$.

  Nous avons $f(x_1) = f(1,2) = -21 < f(1,4) = -25$. Par conséquent, $x_2 = (1,4)$ et à nouveau nous allongeons la longueur de pas $\Delta_2 = \tau^{m_0^+}\Delta_1 = 4$ (voir la figure 2.11).

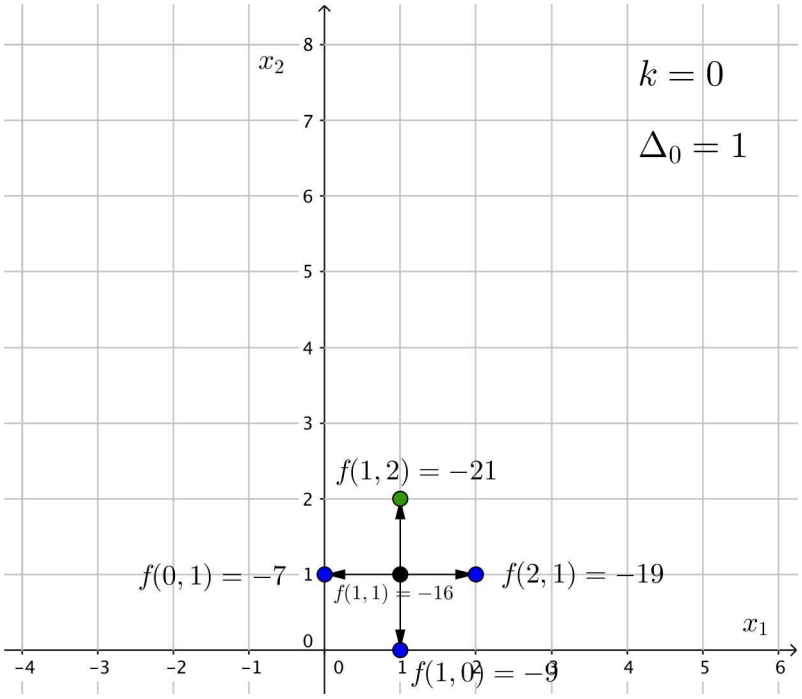


FIGURE 2.10 – Eléments dans le voisinage de x_0 sondés suivant quatre directions avec $\Delta_0 = 1$: le test est un *succès*.

- $\underline{k = 2}$:
  - $f((1,4) + (4,0)) = f(5,4) = -1$;
  - $f((1,4) + (0,4)) = f(1,8) = -9$;
  - $f((1,4) + (-4,0)) = f(-3,4) = 47$;
  - $f((1,4) + (0,-4)) = f(1,0) = -9$.

  Etant donné que nous n'obtenons pas une meilleure valeur de la fonction, nous allons réduire la longueur de pas $\Delta_3 = \tau^{\bar{m_0}} \Delta_2 = 2$ et garder la valeur de l'itéré courant $x_3 = x_2$; le test est un *échec* (voir la figure 2.12).

- $\underline{k = 3}$:
  - $f((1,4) + (2,0)) = f(3,4) = -25$;
  - $f((1,4) + (0,2)) = f(1,6) = -21$;
  - $f((1,4) + (-2,0)) = f(-1,4) = -1$;
  - $f((1,4) + (0,-2)) = f(1,2) = -21$.

  Comme précédemment, nous n'obtenons pas une meilleure valeur et par conséquent, nous réduisons la longueur de pas pour l'itéré suivant $\Delta_4 = \tau^{\bar{m_0}} \Delta_3 = 1$. De la même manière que précédemment, nous gardons

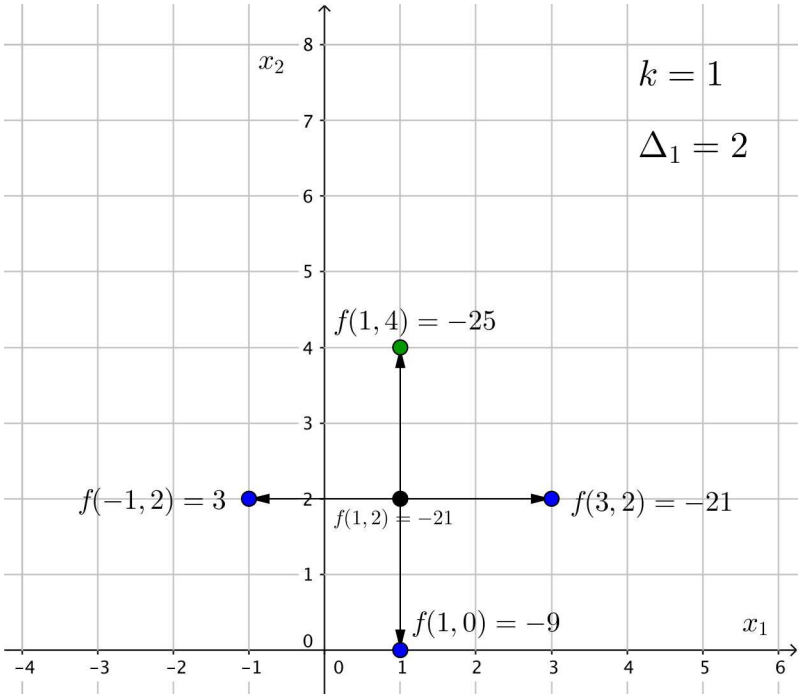


FIGURE 2.11 – Elements dans le voisinage de $x_1 = (1, 2)$ sondés suivant quatre directions avec $\Delta_1 = 2$: le test est un *succès*.

la même valeur de l'itéré courant $x_4 = x_3$ (voir la figure 2.13).

- $\underline{k = 4}$:
  - $f((1,4) + (1,0)) = f(2,4) = -28$;
  - $f((1,4) + (0,1)) = f(1,5) = -24$;
  - $f((1,4) + (-1,0)) = f(0,4) = -16$;
  - $f((1,4) + (0,-1)) = f(1,3) = -24$.

  Nous obtenons donc une meilleure valeur de la fonction en $(2, 4)$, dans ce cas-ci nous agrandissons la longueur de pas $\Delta_5 = \tau^{m_0^+} \Delta_4 = 2$ et modifions la valeur de l'itéré c'est-à-dire $x_5 = (2, 4)$ (voir la figure 2.14).

En réalité, nous avons déjà atteint le minimum de la fonction et par conséquent, l'algorithme va continuer à diminuer la longueur de pas, jusqu'à atteindre un seuil 10^{-6} (correspond à `MeshTol` dans les options MATLAB). Nous n'allons pas détailler les étapes qui suivent. Cela signifiera expliciter les 20 prochaines itérations, où on ne changera pas d'itéré courant, mais bien de diminuer à chaque fois la longueur de pas d'un facteur $\tau^{m_0^-} = 2^{-1}$ et par conséquent, la longueur de pas atteindra $9,5367.10^{-7} = 2^{-20}$.

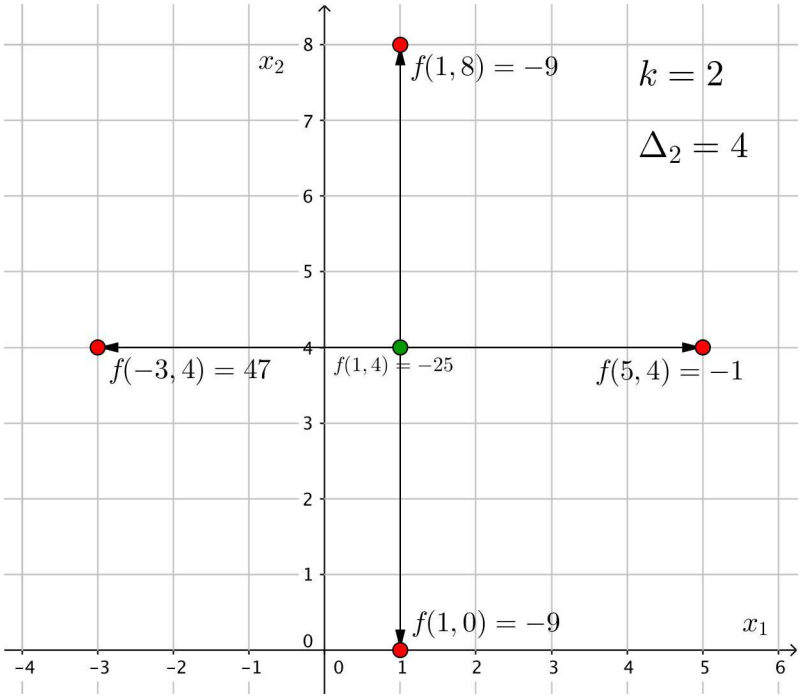


FIGURE 2.12 – Eléments dans le voisinage de $x_2 = (1, 4)$ sondés suivant quatre directions avec $\Delta_2 = 4$: le test est un *échec*.

2.3.6 Fonction MATLAB

La fonction MATLAB [15] de l'algorithme Pattern Search se décrit de la manière suivante

```
[x,fval,exitflag,output] = patternsearch(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options
```

Inputs

- `fun` est la fonction à minimiser ;
- `x0` est le point initial ;
- `A` et `b` représentent les contraintes d'inégalités linéaires de la forme `A*x <= b` où `A` encode les m inégalités linéaires, `b` est un vecteur de taille m et `x` est un vecteur de taille n (nombre de variables) ;
- `Aeq` et `beq` possèdent les même rôles que `A` et `b` respectivement sauf qu'ils interviennent dans le cas des contraintes d'égalités de la forme `Aeq*x = beq` (les dimensions sont identiques) ;
- `lb` donne les bords inférieurs tels que, pour tout `i`, `x(i) >= lb` ;
- `ub` donne les bords supérieurs tels que, pour tout `i`, `x(i) <= ub` ;

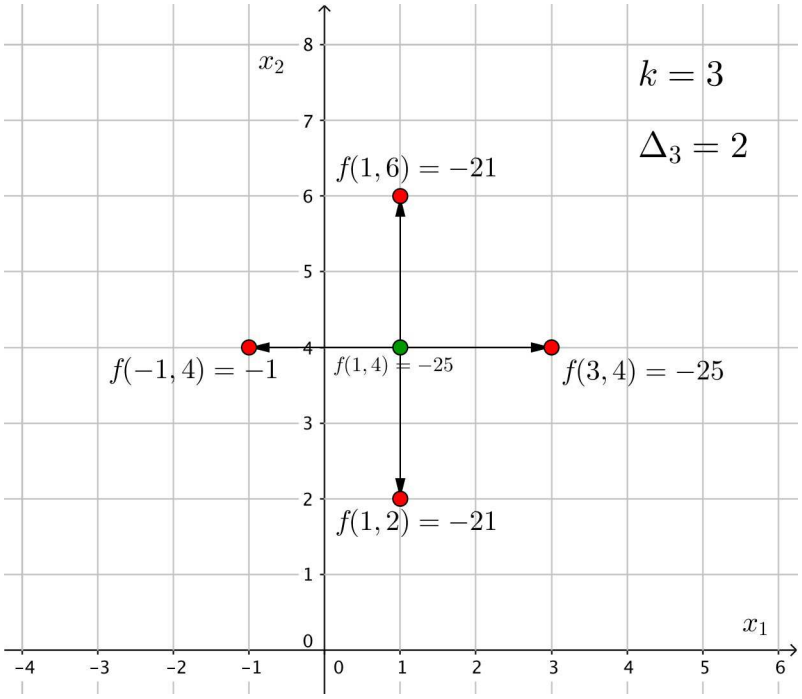


FIGURE 2.13 – Eléments dans le voisinage de $x_3 = (1, 4)$ sondés suivant quatre directions avec $\Delta_3 = 2$: le test est un *échec*.

- `nonlcon` est une fonction qui accepte comme argument `x` et retourne deux tableaux : `c` et `ceq` qui représentent les contraintes d'inégalités et d'égalités non linéaires respectivement en `x` tels que `c(x) <= 0` et `ceq(x) = 0` ;
- `options` permet de spécifier certaines précisions quant à l'optimisation cela peut être :
  - `AccelerateMesh` permet d'accélérer la convergence lorsque l'algorithme est proche d'un minimum (contraction d'un facteur $\frac{1}{2}$. Par défaut la valeur est `false` ;
  - `ConstraintTolerance` donne la tolérance des contraintes. La valeur par défaut est 10^{-6} ;
  - `Display` montre certains caractères du processus tels que
    - `final` montre uniquement le résultat final (valeur par défaut) ;
    - `off` ou `none` ne montre pas de résultat ;
    - `iter` montre les résultats à chaque itération ;
    - `diagnose`.
  - `FunctionTolerance` permet d'arrêter les itérations lorsque le changement au niveau des valeurs de la fonction est inférieur à `FunctionTolerance` et la taille de la grille de valeurs est plus petite que `StepTolerance`. La valeur par défaut est 10^{-6} ;

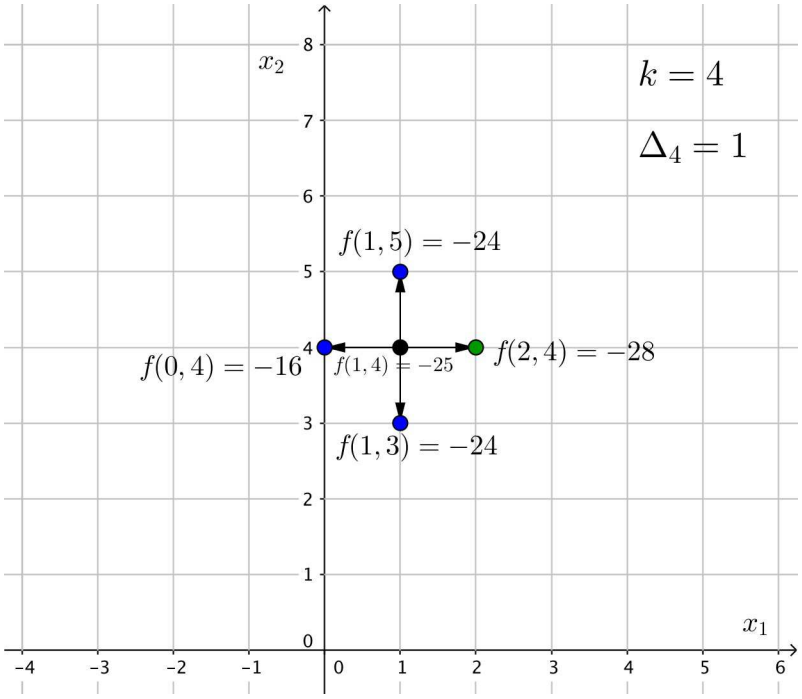


FIGURE 2.14 – Eléments dans le voisinage de $x_4 = (2, 4)$ sondés suivant quatre directions avec $\Delta_4 = 1$: le test est une *réussite*.

- `InitialMeshSize` donne la taille initiale de la grille de valeurs (1 par défaut) ;
- `MaxFunctionEvaluations` précise le nombre maximum d'évaluations de la fonction. La valeur définie par défaut est `2000*numberOfVariables` ;
- `MaxIterations` donne le nombre maximum d'itérations. La valeur définie par défaut est `100*numberOfVariables` ;
- `MaxTime` est le temps maximal alloué à l'algorithme (`Inf` par défaut) ;
- `MeshContractionFactor` détermine le facteur de contraction au cas où le test de sondage est un échec (0,5 par défaut) ;
- `MeshExpansionFactor` détermine le facteur d'expansions dans le cas où le test est un succès (2 par défaut) ;
- `MeshTolerance` est la tolérance sur la taille de la grille de valeurs (10^{-6} est la valeur par défaut) ;
- `OutputFcn` donne une ou plusieurs fonctions déterminées par l'utilisateur dont l'algorithme va faire appel à chaque itération. Le paramètre par défaut est
  `[ ]` ;

- `PlotFcn` permet de donner le graphe de la progression de l'algorithme :
  - `@psplotbestf` montre toutes les meilleures valeurs de la fonction objectif ;
  - `@psplotmeshsize` affiche toutes les longueurs de la grille de valeurs ;
  - `@psplotfuncount` montre toutes les valeurs du nombre d'évaluation de la fonction ;
  - `@psplotbestx` donne tous les meilleurs points ;
  - `[ ]` est le paramètre par défaut : ne montre rien.
- `PollMethod` spécifie la stratégie de sondage de l'algorithme. Cela peut être :
  - `@GPSPositiveBasis2N` (par défaut) ;
  - `@GPSPositiveBasisNp1` ;
  - `@GSSPositiveBasis2N` ;
  - `@GSSPositiveBasis2p1` ;
  - `@MADSPositiveBasis2N` ;
  - `@MADSPositiveBasis2p1`.
- `PollOrderAlgorithm` détermine l'ordre dans lequel on va sonder la grille de valeurs (`Random` ou `Success` ou `Consecutive` qui est la valeur par défaut) ;
- `ScaleMesh` va mettre à l'échelle de manière automatique ou non les variables (la valeur par défaut est `true`) ;

  `SearchFcn` donne le type de fonction de recherche que l'algorithme va utiliser :
  - `@GPSPositiveBasis2N` ;
  - `@GPSPositiveBasisNp1` ;
  - `@GSSPositiveBasis2N` ;
  - `@GSSPositiveBasis2p1` ;
  - `@MADSPositiveBasis2N` ;
  - `@MADSPositiveBasis2p1` ;
  - `@searchga` ;
  - `@searchlhs` ;
  - `@searchneldermead` ;
  - `[ ]` (par défaut).

- `StepTolerance` indique lorsque les itérations doivent stopper, c'est-à-dire lorsque le changement de position ainsi que la taille de la grille de valeurs sont plus petits que `StepTolerance` (la valeur par défaut est 10^{-6}) ;
- `UseCompletePoll` permet de faire un sondage complet autour du point courant (la valeur par défaut est `false`) ;
- `UseCompleteSearch` permet de faire une recherche complète autour du point courant dans le cas où la méthode de recherche est une méthode de sondage (la valeur par défaut est `false`) ;
- `UseParallel` permet à l'algorithme de calculer la fonction objectif ainsi que les fonctions de contraintes non linéaires en parallèle (la valeur par défaut est `false`) ;
- `UseVectorized` spécifie si les fonctions sont vectorisées (la valeur par défaut est `false`).

Outputs

- `x` est la solution du problème d'optimisation ;
- `fval` est la valeur de la fonction évaluée en la solution calculée ;
- `exitflag` est la raison pour laquelle l'algorithme s'est arrêté :
  - Si `exitflag = 1` : dans le cas où le problème d'optimisation était sans contrainte non linéaire, cela signifie que la longueur de la taille de la grille a atteint le seuil de tolérance. Dans le cas où le problème d'optimisation était avec contraintes non linéaires, la longueur de la mesure de complémentarité (la norme du vecteur dont les éléments sont $c_i\lambda_i$ où les c_i sont les contraintes d'inégalité non linéaires et les λ_i sont les multiplicateurs de Lagrange) est plus petite que `sqrt(ConstraintTolerance)`, le sous-problème est résolu en dépassant la tolérance de la taille de la grille de valeur (`MeshTolerance`) et la violation de contrainte est plus petite que `ConstraintTolerance` ;
  - Si `exitflag = 2`, alors le changement en `x` et la taille de la grille sont tous deux plus petits que le seuil et la violation de contrainte est plus petite que `ConstraintTolerance` ;
  - Si `exitflag = 3`, de la même manière que dans le cas où `exitflag = 2`, cela signifie que le changement en `fval` et la taille de la grille sont tous deux plus petits que la tolérance définie préalablement et la violation de contrainte est plus petite que `ConstraintTolerance` ;
  - Si `exitflag = 4`, la longueur du pas est plus petite que la précision machine (soit $2,2204.10^{-16}$) et la violation de contrainte est plus petite que `ConstraintTolerance` ;

  - Si `exitflag = 0`, le nombre maximum d'évaluations de la fonction ou le nombre maximum d'itérations sont atteints ;
  - Si `exitflag = -1`, l'optimisation s'est arrêtée par une fonction résultante ou un graphe de la fonction ;
  - Si `exitflag = -2`, il n'y a pas eu de point trouvé.
- `output` fournit des informations sur le processus qui s'est déroulé comme :
  - `function`, donne la fonction objectif ;
  - `problemtype`, précise le type de problème considéré comme
    * `unconstrained` (sans contrainte) ;
    * `boundconstraints` (avec contraintes aux bords) ;
    * `linearconstraints` (avec contraintes linéaires) ;
    * `nonlinearconstr` (avec contraintes non linéaires).
  - `pollmethod` spécifie la technique de sondage utilisée ;
  - `searchmethod` spécifie la technique de recherche si jamais cela devait se préciser ;
  - `iterations` donne le nombre d'itérations ;
  - `funcCount` donne le nombre d'évaluations de la fonction ;
  - `meshsize` donne la taille de la grille en `x` ;
  - `maxconstraint` donne la violation maximale de la contrainte si jamais cela devait se préciser ;
  - `rngstate` donne l'état du générateur de nombres aléatoires juste avant que l'algorithme ne démarre.
  - `message` fournit le message de sortie (la raison pour laquelle l'algorithme arrête d'itérer).

La troisième et dernière partie de ce mémoire consiste à appliquer ces différents aspects théoriques à des images concrètes. Cela permet de visualiser les points faibles et points forts des différents algorithmes utilisés.

Troisième partie

Application à l'astrophotographie

Chapitre 1

Recherches et réflexions sur le recalage

1.1 Problème posé

Il existe plusieurs méthodes pour recaler deux images. Celle que nous avons étudié s'intéresse à l'optimisation, en particulier à la maximisation de la mesure de similarité. En effet, considérons la mesure de similarité $f(x)$ où $x \in \mathbb{R}^3$ est un vecteur représentant les paramètres de la transformation géométrique (angle de rotation, pixels de déplacements verticaux et horizontaux). Nous cherchons donc à résoudre le problème d'optimisation

$$\max_{x \in \mathbb{R}^3} f(x).$$

Etant donné que les algorithmes d'optimisation implémentés dans MATLAB consistent à minimiser une fonction objectif, il suffit d'inverser le signe de la fonction de départ afin de trouver le maximum. Le problème devient alors

$$\min_{x \in \mathbb{R}^3} - f(x).$$

Néanmoins, par souci de rapidité étant donné que les images sont de grandes résolutions, il est intéressant de la réduire. Par choix, nous réduisons la hauteur de l'image à 100 pixels. De cette manière, le temps d'exécution des algorithmes est fixé. Il s'agit donc d'une méthode qui utilise en quelque sorte la multirésolution ainsi que le recalage basé sur l'optimisation abordés dans les points théoriques précédents (section 1.6). La différence est que, ici, nous ne diminuons pas pendant un certain nombre d'itérations la résolution mais bien qu'une seule fois.

1.2 Stratégie employée

1.2.1 Choix de l'image

Comme annoncé à la section 1.4, les images à traiter en astrophotographie ne sont pas toutes de la même nature. En effet, un objet ponctuel (tel que la Lune) n'aura pas les mêmes caractéristiques qu'un objet plus diffus (comme une galaxie ou une nébuleuse). La mesure de similarité aura probablement une influence différente qu'une autre ou bien, il faudra utiliser une autre stratégie. C'est pour cela que lors de l'étude de cas, nous analyserons plusieurs types d'objets qui sont les suivants :

1. Logo de l'Université de Namur (taille 400×361) ;
2. Paysage alsacien (taille 768×1024) ;
3. La Lune (taille 1024×773) ;
4. La Nébuleuse d'Orion (taille 773×1024).

Dans les fichiers MATLAB, l'utilisateur peut choisir le type d'image à traiter ; il s'agira de l'image de référence dénotée dans les codes par `fixed`.

```
1  choix_IMG = 1;
2
3  if (choix_IMG == 1)
4
5      fixed = im2double(rgb2gray(imread('UNamur.JPG')));
6
7  elseif (choix_IMG == 2)
8
9      fixed = im2double(rgb2gray(imread('Alsace.JPG')));
10
11 elseif (choix_IMG == 3)
12
13     fixed = im2double(rgb2gray(imread('Lune.JPG')));
14
15 elseif (choix_IMG == 4)
16
17     fixed = im2double(rgb2gray(imread('Nebuleuse.JPG')));
18
19 end;
```

Pour toutes ces images, nous l'avons tournée de 10 degrés vers la gauche uniquement via un logiciel de modification d'images (« Photothèque » sur Mac) afin d'avoir une idée de la qualité du recalage (en plus de la valeur du $PSNR$ abordée précédemment). Par conséquent, l'image de détection `moving` (celle qui sera modifiée pour obtenir un recalage), est chargée en fonction de l'image choisie par l'utilisateur via la commande

```
moving = charger(fixed, choix_IMG);
```

Remarque 1.2.1. *Les images à recaler ne sont pas forcément de mêmes tailles. D'ailleurs, ce n'est pas le cas ici. Par conséquent, lorsque nous devons charger les images, pour pouvoir appliquer les mesures de similarité, nous devons faire en sorte que les images soient de même taille. C'est pour cela que dans la fonction* `charger(X,choix)`*, nous initialisons une nouvelle image contenant uniquement des zéros de taille égale à celle de référence contenant l'image de détection. Par conséquent, la nouvelle image de détection se situera dans le coin supérieur gauche. Cela ne pose pas de problème car la fonction d'optimisation va recaler en fonction de ces nouveaux paramètres de translation. Nous sommes dans un problème de transformation géométrique, il ne faut donc pas agrandir ni rétrécir l'image de sorte à garder les mêmes dimensions. L'encadré suivant reprend les différentes instructions illustrant cette remarque.*

```
1  [m, n] = size(X); % Il faut que les dimensions soient ...
       identiques
2  Y = zeros([m,n]);
3
4  if (choix == 1)
5          innput = im2double(rgb2gray(imread('UNamur10.jpg')));
6  (...)
7  end
8
9  [o, p] = size(innput);
10
11 for i = 1 : o
12         for j = 1 : p
13             Y(i,j) = innput(i,j);
14         end
15 end
```

Ensuite, comme signalé précédemment, par souci de rapidité de l'algorithme, nous diminuons la résolution des images originales avec une hauteur de 100 pixels ; les nouvelles variables sont alors `fixed_degr` et `moving_degr`. Dans le fichier principal, nous appelons la fonction

```
1  fixed_degr = degrade(fixed);
2  moving_degr = degrade(moving);
```

Cette dernière se décrit de la manière suivante

```
1  function Y = degrade(X)
2      [m n] = size(X);
3
4      m_new = 100;
```

```
5       n_new = (m_new/m)*n;
6
7       Y = imresize(X,[m_new, n_new]);
8   end
```

Au vu de la remarque 1.2.1, les paramètres de translation seront une adaptation de cette mise à l'échelle.

1.2.2 Condition initiale

Dans la plupart des algorithmes d'optimisation, nous avons besoin d'une condition initiale, par choix nous la choisissons à $(0, 0, 0)$. En effet, ce choix est motivé par le fait qu'au mieux l'image de détection n'est pas une copie modifiée de l'image de référence. D'autre part, nous aurions pu choisir une autre condition ; pour ce faire, il suffit de modifier la commande

```
x_0 = [0 , 0 , 0];
```

1.2.3 Choix de la mesure de similarité

Après avoir établi, les bases concernant le choix de l'image à recaler et la condition initiale, nous pouvons choisir une mesure de similarité :

1. Coefficient de corrélation de Pearson ;
2. Minimum ratio ;
3. Rho de Spearman ;
4. Structural Similarity.

Pour ce faire, l'utilisateur choisit la mesure de similarité grâce à la commande dans le fichier principal

```
1 choix_MESURE = 1;
2
3 fun = similarite(fixed_degr, moving_degr, choix_MESURE);
```

La fonction permet de considérer la mesure de similarité choisie par l'utilisateur. Celle-ci est décrite par les lignes de codes suivantes

```
1 function mesure = similarite(X, Y, choix)
2
3     if (choix == 1)
4         mesure = @pearson ;
5
6     elseif (choix == 2)
```

```
7           mesure = @minimumRatio ;
8
9       elseif (choix == 3)
10          mesure = @spearmanRho ;
11
12      elseif (choix == 4)
13          mesure = @ssimMoi ;
14
15  end
```

1.2.4 Choix de la méthode d'optimisation

La dernière étape avant d'avoir des résultats est de choisir la méthode d'optimisation qui va permettre de maximiser la mesure de similarité choisie. Les méthodes que nous proposons dans ce mémoire sont les suivantes (leurs algorithmes sont décrites dans la partie théorique 2) :

1. Méthode de Nelder-Mead ;
2. Algorithme génétique ;
3. Pattern Search.

Le fichier incluant les différentes méthodes sont inscrites dans le fichier fctOpti.m :

```
1  function [param_Opti iter] = fctOpti(fun, X, Y, x_0, choix)
2
3      if (choix == 1)
4
5          [param_Opti fval exitval iter]= ...
               fminsearch(@(param)-fun(X,Y,param),x_0);
6
7      elseif (choix == 2)
8
9          [param_Opti fval exitval iter] = ...
               ga(@(param)-fun(X,Y,param), 3, [],[],[],[],[]);
10
11     elseif (choix == 3)
12
13         [param_Opti fval exitval iter] = ...
               patternsearch(@(param)-fun(X,Y,param), ...
               x_0,[],[],[],[],[]);
14
15     end
16
17 end
```

Ensuite, dans le fichier principal Opti.m, les commandes du choix de la méthode d'optimisation choisie sont

```
choix_METH = 1;

[param_Opti1 iter] = fctOpti(fun, fixed_degr, moving_degr, ...
    x_0, choix_METH)

param_Opti2 = fminsearch(@(param)-fun(fixed_degr, ...
    moving_degr, param),param_Opti1)
```

Remarquons que nous effectuons deux fois la recherche de maximum ; ceci est dans l'intérêt de trouver une meilleure solution que celle obtenue avec la première méthode. La différence est que nous utilisons d'office la méthode de Nelder-Mead dans la deuxième optimisation. Cependant, comme nous le verrons dans la plupart des cas étudiés, nous n'obtenons pas forcément une valeur très distincte de la première.

1.2.5 Application des paramètres obtenus

Après avoir trouvé les paramètres de recalage (angle de rotation, déplacements verticaux et horizontaux), nous pouvons les appliquer à l'image de détection d'origine ; sans avoir réduit la résolution. Etant donné que les paramètres trouvés sont propres aux images incluses dans les méthodes d'optimisation (c'est-à-dire les images dont la résolution a été réduite), il faut remettre à l'échelle ceux-ci afin d'obtenir un recalage adapté à la résolution de départ. C'est pour cela que

- nous enregistrons dans des variables la taille de l'image de référence ;
- nous calculons le ratio de diminution de la résolution qui a été effectuée (on sait qu'elle a été diminuée de sorte que nous ayons 100 pixels de hauteur) ;
- nous multiplions ce ratio par les paramètres de recalage de déplacements horizontaux et verticaux trouvés précédemment.

Remarque 1.2.2. *Nous ne multiplions pas le paramètre de rotation étant donné que l'angle est intrinsèque dans ce type de recalage.*

Par conséquent, les lignes de code sont les suivantes

```
[m n] = size(fixed);
m_r = m/100;

recal = geometricT([param_Opti2(1) param_Opti2(2)*m_r ...
    param_Opti2(3)*m_r], moving);
```

Remarque 1.2.3. *Une particularité de l'utilisation de la fonction* `geometricT` *est qu'elle utilise elle-même la fonction* `imrotate` *propre à MATLAB. Il existe différentes options pour tourner une image notamment si nous choisissons de garder les dimensions de l'image de départ (*`'crop'`*) ou bien que celle-ci soit tournée sans restreindre les dimensions (la fonction le fait par défaut). Or dans notre cas, nous avons utilisé cette dernière option afin d'éviter de perdre de l'information en restreignant les dimensions de l'image. Par conséquent, les dimensions entre l'image de référence et l'image déformée après l'optimisation ne sont plus identiques. C'est pour cela que nous faisons appel à la fonction* `redim` *qui prend en argument l'image à redéfinir ainsi que les dimensions de l'image. Visuellement, cela va enlever les bords inutiles apparus lors des différentes rotations. Les lignes de codes ci-dessous illustrent cette remarque.*

```
1 function Y = redim(X,dim)
2     Y = zeros(dim);
3
4     for i = 1 : dim(1)
5         for j = 1 : dim(2)
6             Y(i,j) = Y(i,j) + X(i,j);
7         end;
8     end;
9 end
```

Le chapitre suivant reprend les différents résultats obtenus lors des analyses de méthodes d'optimisation combinées avec des mesures de similarité.

Chapitre 2

Résultats obtenus

Pour toutes les images, l'image de détection est celle de référence uniquement tournée de 10 degrés vers la droite ; il n'y a donc pas de déplacements verticaux et horizontaux. Par conséquent, le résultat du recalage devrait nous donner les paramètres

$$[\theta, h, k] = [10, 0, 0],$$

puisque la fonction `imrotate` tourne l'image vers la gauche si l'angle pris en argument est positif ; en d'autres termes, le sens positif est dans le sens anti-horlogique.

2.1 Logo de l'UNamur

(a) Image de référence (b) Image de détection (c) Superposition des images sans recalage

FIGURE 2.1 – Logo de l'UNamur (400×361).

Les images à recaler sont reprises à la figure 2.1 ainsi qu'un aperçu de la superposition des images. À l'oeil nu, il est aisé de remarquer la différence entre les deux images. Evidemment, dans une situation plus concrète, la différence sera plus faible voire plus difficilement visible.

Remarque 2.1.1. *Les couleurs vertes et magentas dans la superposition des images représentent les différences d'intensité. Si elles sont toutes identiques, alors nous avons une image dans les nuances de gris.*

Les sections suivantes reprennent les valeurs de toute une étude concernant cette image. En effet, nous allons combiner les différentes mesures de similarité que nous avons abordées avec les méthodes d'optimisation et analyser ce que nous obtenons.

2.1.1 Mesure de similarité : Coefficient de Pearson

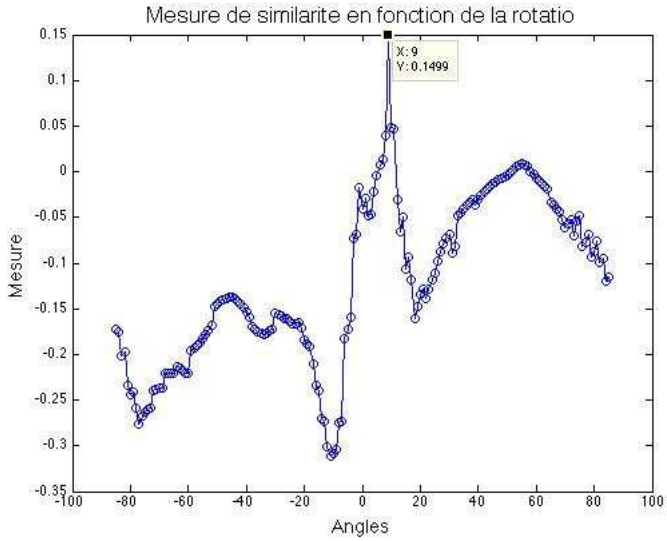
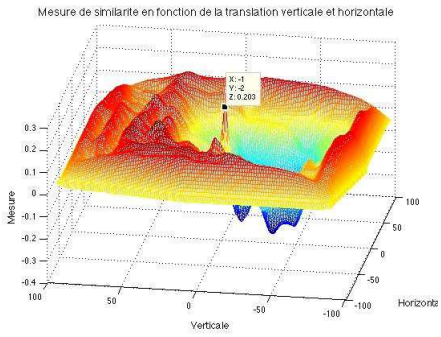


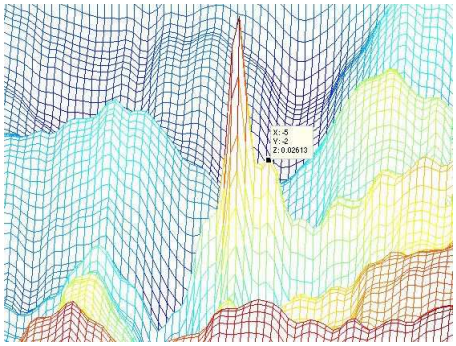
FIGURE 2.2 – Mesure de similarité (coefficient de Pearson) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.

Afin d'avoir une idée de l'allure de la mesure de similarité, nous représentons d'une part cette dernière en fonction de l'angle de rotation (figure 2.2) pour h et k fixés ; d'autre part la mesure de similarité en fonction des déplacements verticaux et horizontaux h et k respectivement pour un θ fixé à 10 (figure 2.3).

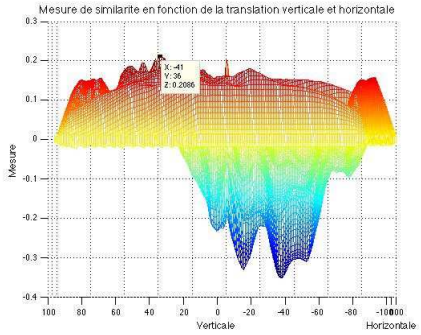
Au vu de la figure 2.2, le paramètre de rotation maximum ne correspond pas exactement à celui attendu (c'est-à-dire 10). En effet, cela est en partie dû à la discrétisation de l'intervalle de paramètres de la fonction. Par conséquent, plus nous allons diminuer le pas de discrétisation, plus nous obtiendrons une meilleure approximation de la fonction (mais pas forcément une fonction plus lisse et plus régulière). Cela permet tout de même de nous donner une idée de la régularité de la mesure de similarité ainsi que le niveau de difficulté à atteindre le maximum. En effet, nous pouvons déjà constater la présence de



(a) Mesure en fonction des déplacements verticaux et horizontaux h et k respectivement.



(b) Zoom sur le maximum de la fonction.



(c) Vue latérale.

FIGURE 2.3 – Mesure de similarité (coefficient de Pearson) en fonction des paramètres de translation.

maxima locaux et sont donc susceptibles d'attirer l'algorithme d'optimisation vers l'un d'entre eux.

Concernant la figure 2.3, nous constatons que la fonction n'est pas plus facile à optimiser ni à étudier. En effet, nous pouvons remarquer la présence de plusieurs maxima locaux dont un n'étant pas celui qui correspond à un bon recalage (vue latérale) et donne bel et bien une mesure de similarité plus élevée. Cela peut donc conduire l'algorithme à être attiré vers ce maximum alors qu'il ne nous intéresse pas. Or celui qui nous intéresse se trouve sur un pic représenté au milieu (en $(-1; -2)$). De plus, aux environs de ce point intéressant, il y a encore des maxima locaux « dangereux » pour les algorithmes d'optimisation.

Par conséquent, cet exemple qui se voulait simple et académique contient de nombreux cas particuliers. Les différents résultats sont repris dans les tableaux 2.1 et 2.2 à la page 108.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	7,54	123	602
Algorithme génétique	96, 63	145	7300
Pattern Search	3,37	62	300

TABLE 2.1 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour le coefficient de Pearson et le logo de l'UNamur.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	4,79	4,79	$[0; 0; 2, 5.10^{-4}]$	$[0; 0; 2, 6.10^{-4}]$	1	1
Algorithme génétique		2,83	$[6, 49; -38, 69; 37, 58]$	$[6, 98; -38, 45; 37, 56]$	$0, 35$	$0, 3$
Pattern Search		5,49	$[10, 67; -1; -2]$	$[10.67 - 1 - 2]$	$6, 61.10^{-2}$	$6, 61.10^{-2}$

TABLE 2.2 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour le coefficient de Pearson et le logo de l'UNamur.

Dans le tableau 2.1, la méthode d'optimisation Pattern Search possède son avantage dans tous les cas : tant au niveau du temps d'exécution que pour le nombre d'évaluation de la fonction.

Concernant le tableau 2.2, la méthode de Pattern Search montre à nouveau de bons résultats par rapport aux autres algorithmes : les paramètres sont ceux recherchés et la valeur du $PSNR$ augmente. Par contre, ce dernier diminue pour l'Algorithme Génétique . D'ailleurs ce dernier plonge vers le maximum qui ne nous intéresse pas comme nous l'avions souligné dans le paragraphe précédent (soit le point de coordonnées $(-41; 36)$ représenté à la figure 2.3). Le même comportement se fait au niveau du calcul de l'angle : il trouve un maximum local dans ce cas-ci.

2.1.2 Mesure de similarité : Minimum Ratio

Pour les mesures qui suivent, nous allons procéder au même type d'analyse que pour le coefficient de Pearson.

À la figure 2.4, nous pouvons voir qu'il est extrêmement difficile de connaître la position du maximum. Par conséquent, les algorithmes risquent également de stagner autour de ces valeurs sans trouver le vrai maximum parmi ces points.

Concernant les déplacements verticaux et horizontaux à la figure 2.5, lorsque nous analysons la vue latérale de la fonction, cette dernière est également plate. Le maximum va donc également être difficile à trouver. Par conséquent, à priori, cette mesure nous dévoile un de ses points faibles.

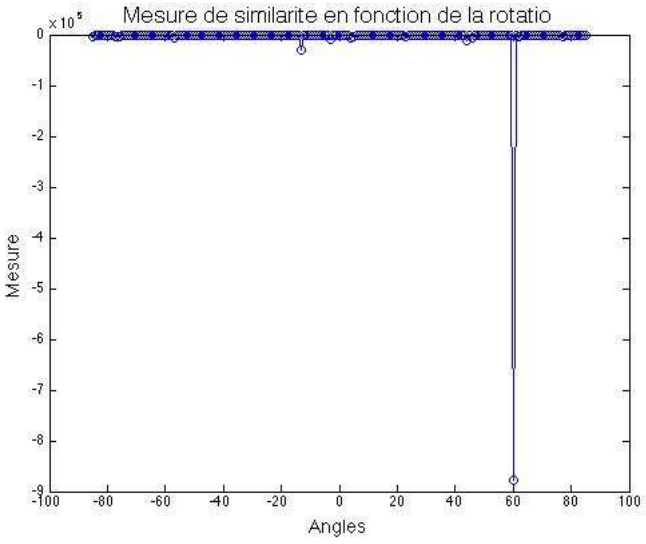
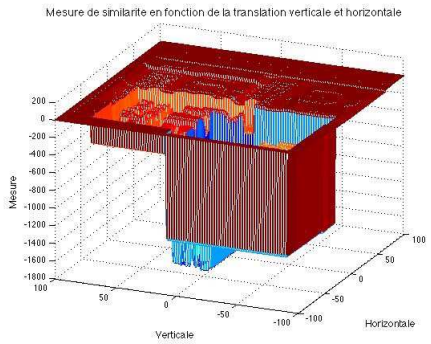
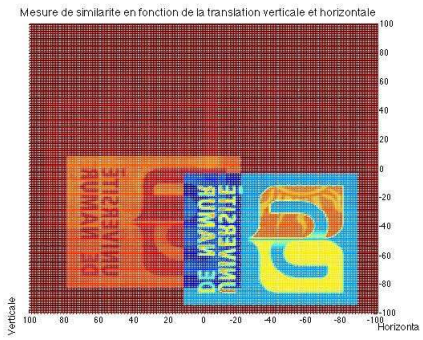


FIGURE 2.4 – Mesure de similarité (Minimum Ratio) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.

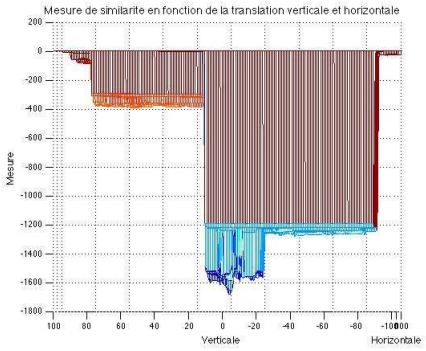
Les résultats sont donnés dans les tableaux 2.3 et 2.4 à la page 111.



(a) Mesure en fonction des déplacements verticaux et horizontaux h et k respectivement.



(b) Vue du dessus.



(c) Vue latérale.

FIGURE 2.5 – Mesure de similarité (Minimum Ratio) en fonction des paramètres de translation.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	7,8	130	602
Algorithme génétique	117,7219	213	10700
Pattern Search	2,89	28	159

TABLE 2.3 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour le Minimum Ratio et le logo de l'UNamur.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	4,79	4,79	$[0; 0; 0]$	$[0; 0; 0]$	1	1
Algorithme génétique		4,52	$[0,62; 13,09; 14,77]$	$[0,62; 13; 14,75]$	$0,94$	$0,94$
Pattern Search		4,79	$[0; 2; 3]$	$[0; 2; 3]$	1	1

TABLE 2.4 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour le Minimum Ratio et le logo de l'UNamur.

Sans surprise, les résultats ne sont pas intéressants du tout. Nous n'allons donc par retenir cette mesure-ci dans le cas des images planétaires (l'image du logo présente environ les mêmes caractéristiques qu'une planète) ; quoi qu'il en soit, le problème est mal posé.

2.1.3 Mesure de similarité : Rho de Spearman

La figure 2.6 nous indique que la fonction est bien définie mais qu'elle possède également des maxima locaux.

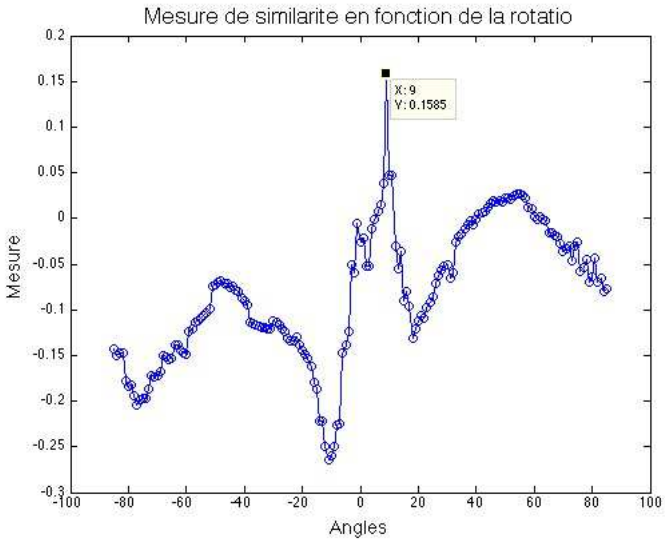


FIGURE 2.6 – Mesure de similarité (Rho de Spearman) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.

Remarque 2.1.2. *Nous n'avons pas inclus les graphes contenant les différents déplacement verticaux et horizontaux car le temps de calculs est excessivement long. En effet, pour établir cette figure, nous devons calculer toutes les valeurs des mesures de similarité pour tous les déplacements verticaux et horizontaux possibles. De plus, comme nous l'avons souligné dans la partie théorique de ce mémoire, la complexité de cette mesure est de l'ordre de* $n \log n$*. La manipulation d'images rend les calculs encore plus longs. En effet, en prenant l'image du logo de l'université de Namur, et en réduisant la résolution, nous arrivons à une matrice de taille* 100×96 *ce qui signifie* 9600^2 *comparaisons en considérant l'image de déformation.*

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	431,75	135	602
Algorithme génétique	$1,05.10^3$	55	2800
Pattern Search	490,83	136	646

TABLE 2.5 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour le ρ de Pearson et le logo de l'UNamur.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	4,79	4,79	$[0; 0,13.10^{-3}; 0,08.10^{-3}]$	$[0; 0,13.10^{-3}; 0,08.10^{-3}]$	1	1
Algorithme génétique		1,65	$[0,06; 91,81; -41,41]$	$[0,06; 91,81; -41,41]$	0,99	0,99
Pattern Search		5,51	$[10,75; -1; -2,49]$	$[10,75; -1; -2,49]$	0,08	0,08

TABLE 2.6 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour le ρ de Spearman et le logo de l'UNamur.

Un inconvénient reflété dans ces résultats est le temps de calculs des paramètres du recalage (la remarque précédente s'y applique également mais pas à la même échelle de temps), en particulier pour l'algorithme génétique. Nous pouvons donc constater l'ampleur de la complexité dans le traitement d'images numériques. Ensuite, au vu des résultats, la méthode Pattern Search donne de bons résultats. Cependant, la précision n'est pas aussi bonne que pour le coefficient de Pearson et par souci de rapidité, nous ne retiendrons pas cette mesure pour la seconde partie de l'analyse.

2.1.4 Mesure de similarité : SSIM

De la même manière que précédemment, considérons les figures 2.7 et 2.8 qui permettent d'avoir une idée des allures des différentes fonctions et de savoir si le problème est bien posé.

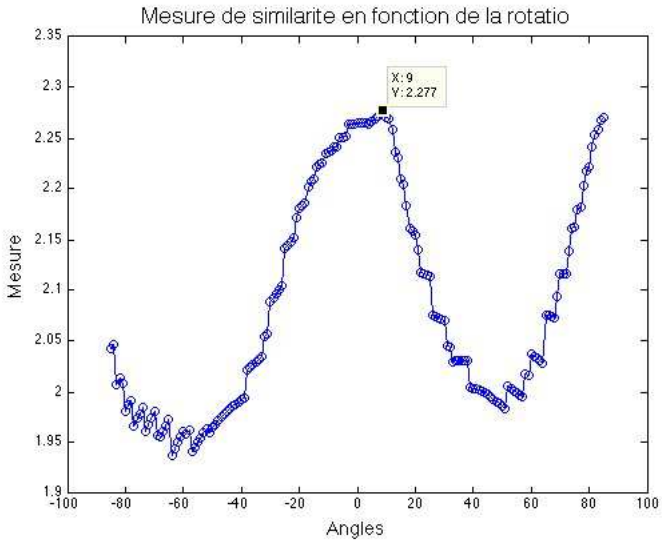
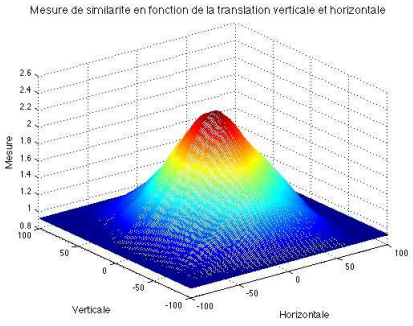


FIGURE 2.7 – Mesure de similarité (SSIM) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.

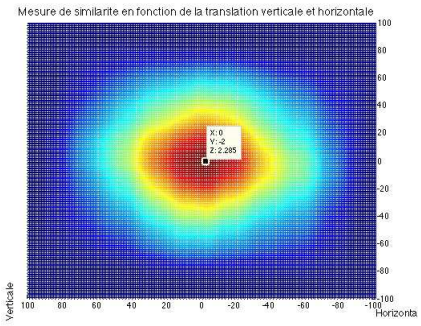
Concernant l'allure de la fonction représentée à la figure 2.7 lorsque θ varie, celle-ci présente des maxima locaux également. Par cette observation, nous pouvons penser que certains algorithmes s'arrêteraient en ces endroits, et ne pas continuer vers la valeur de 10 degrés.

Remarque 2.1.3. *Le maximum indiqué à la figure 2.7 n'est pas exactement dix étant donné que l'allure de la fonction dépend de la manière dont nous avons discrétisé l'intervalle du paramètre de rotation.*

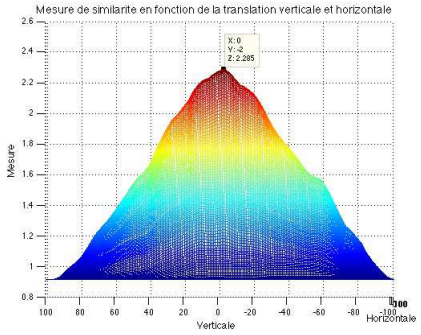
Par rapport à la figure 2.8 à la page 115 représentant la mesure de similarité en fonction des déplacements verticaux et horizontaux, nous avons un seul



(a) Mesure en fonction des déplacements verticaux et horizontaux h et k respectivement.



(b) Vue du dessus.



(c) Vue latérale.

FIGURE 2.8 – Mesure de similarité (SSIM) en fonction des paramètres de translation.

maximum bien mis en valeur. Les résultats de la qualité du recalage pour les différentes méthodes d'optimisation sont repris dans les tableaux 2.7 et 2.8 à la page 116.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	3,15	123	221
Algorithme génétique	104,55	148	7450
Pattern Search	4,93	66	335

TABLE 2.7 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour SSIM et le logo de l'UNamur.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	4,79	4,79	$[-0,63; 0.5; 0,96]$	$[-0,63; 0,5; 0,96]$	$1,06$	$1,06$
Algorithme génétique		5,62	$[11,35; -0,69; -1,83]$	$[11,25; -0,70; -2]$	$0,14$	$0,13$
Pattern Search		4,81	$[-1,89; -1,63; -1]$	$[-1,89; -1,63; -1]$	$1,19$	$1,19$

TABLE 2.8 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour SSIM et le logo de l'UNamur.

À présent, analysons les résultats. Dans le premier tableau, l'algorithme génétique est bien plus lent et plus couteux que les deux autres. Cependant, en regardant le deuxième tableau, il s'agit de l'algorithme donnant les meilleurs résultats. D'une part, la valeur du $PSNR$ augmente le plus (en effet, cela vient du fait que les paramètres estimés sont meilleurs). D'autre part, l'erreur relative est également plus petite. Cela illustre que l'algorithme génétique montre ses points forts quant à cette mesure de similarité. Nous pourrons donc retenir cette combinaison de méthode d'optimisation et de mesure de similarité pour ce type d'image.

Enfin, les paramètres estimés par les deux autres méthodes proviennent des maxima locaux de la fonction comme nous l'avons remarqué au début de l'analyse de cette mesure.

2.2 Paysage d'Alsace

(a) Image de référence.

(b) Image de détection.

(c) Superposition des images sans recalage.

FIGURE 2.9 – Paysage d'Alsace (768×1024).

Les images à recaler sont présentées à la figure 2.9 à la page 117. Le type d'image n'est pas identique au cas étudié précédemment. En effet, nous avons un objet plus « diffus » que dans le cas précédent.

2.2.1 Mesure de similarité : Coefficient de Pearson

Nous pouvons constater dans la figure 2.10, que la fonction possède des maxima locaux dont un global ne correspondant pas à ce qu'on attend. Cela est probablement dû aux caractéristiques de l'image : nous sommes confrontés à un paysage et non plus à un objet bien délimité comme le logo. La mesure de similarité a également un autre comportement (les similarités entre les deux images seront plus difficiles à mesurer). Par conséquent, nous présageons que les solutions des différents algorithmes ne vont pas forcément donner des résultats convaincants.

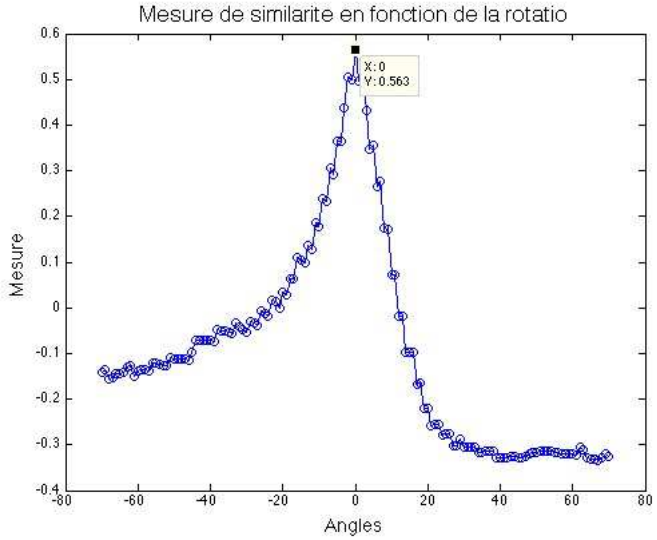
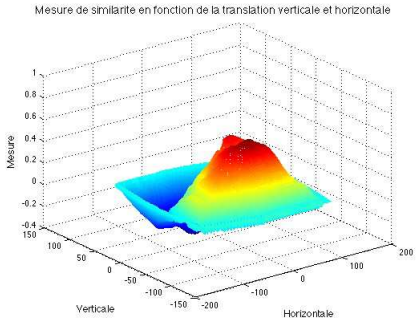


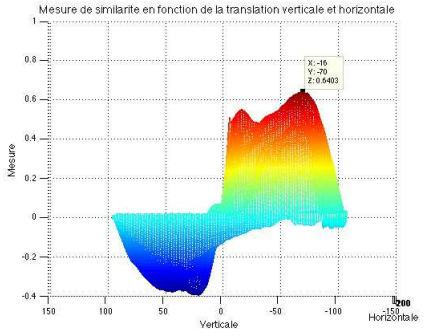
FIGURE 2.10 – Mesure de similarité (coefficient de Pearson) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.

Les commentaires concernant la figure 2.11 à la page 119 sont similaires : le maximum global n'est pas celui recherché et par conséquent, les algorithmes risquent de s'y plonger.

Ensuite, les tableaux 2.9 et 2.10 donnent les résultats des différents recalages. De fait, par le constat au paragraphe précédement, ces résultats ne nous donnent pas de bonnes valeurs. D'ailleurs la valeur du $PSNR$ diminue dans le cas de l'algorithme génétique et l'erreur relative est élevée pour les deux autres.



(a) Mesure en fonction des déplacements verticaux et horizontaux h et k respectivement.



(b) Vue latérale.

FIGURE 2.11 – Mesure de similarité (coefficient de Pearson) en fonction des paramètres de translation.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	3,74	153	277
Algorithme génétique	77,04	137	6900
Pattern Search	6,06	96	469

TABLE 2.9 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour le coefficient de Pearson et le paysage d'Alsace.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	10,07	10,08	$[-0,02; 0,62; -0,51]$	$[-0,02; 0.,62; -0.,51]$	1	1
Algorithme génétique		7,45	$[4,93; -6,37; -63,53]$	$[5,37; -6,53; -63,49]$	$0,51$	$0,46$
Pattern Search		10,41	$[0,25; 12,45; -0,62]$	$[0,2512,45 - 0,62]$	$0,98$	$0,98$

TABLE 2.10 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour le coefficient de Pearson et le paysage d'Alsace.

2.2.2 Mesure de similarité : Minimum Ratio

Dans cette mesure, nous rencontrons le même problème pour la première image : la fonction est plate tant pour la rotation (figure 2.12) que pour la translation(figure 2.13).

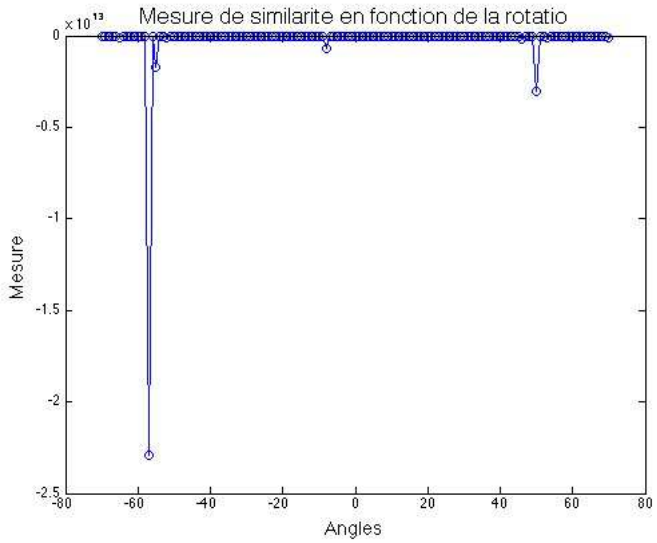
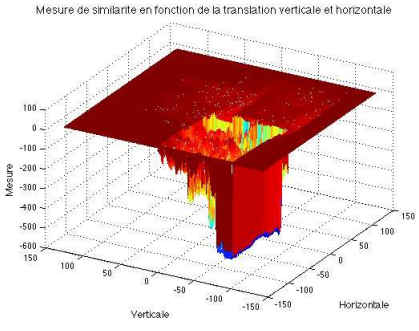
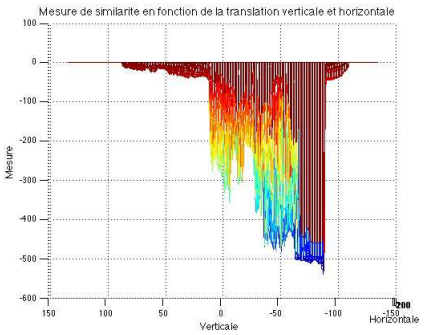


FIGURE 2.12 – Mesure de similarité (Minimum Ratio) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.



(a) Mesure en fonction des déplacements verticaux et horizontaux h et k respectivement.



(b) Vue latérale.

FIGURE 2.13 – Mesure de similarité (Minimum Ratio) en fonction des paramètres de translation.

Comme nous nous y attendions, les résultats (tableaux 2.11 et 2.12) ne sont pas satisfaisants. Par conséquent, par la suite, nous ne retiendrons pas cette mesure car elle n'est intéressante ni pour une image de paysage ni pour une image d'un objet bien délimité.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	7,19	124	603
Algorithme génétique	72,84	119	6000
Pattern Search	6,34	92	472

TABLE 2.11 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour Minimum Ratio et le paysage d'Alsace.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	10,07	10,07	$[0; 0; 0]$	$[0; 0; 0]$	1	1
Algorithme génétique		6,77	$[-0,09; -14,02; 16,55]$	$[-0,10; -14; 16,47]$	1,01	1,01
Pattern Search		7,42	$[0; 22,42; 16,51]$	$[0; 22,42; 16,51]$	1	1

TABLE 2.12 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour Minimum Ratio et un paysage alsacien.

2.2.3 Mesure de similarité : Rho de Spearman

Le graphe représenté à la figure 2.14 montre un maximum ne correspond pas à nos attentes ; il se situe lorsque θ vaut 0.

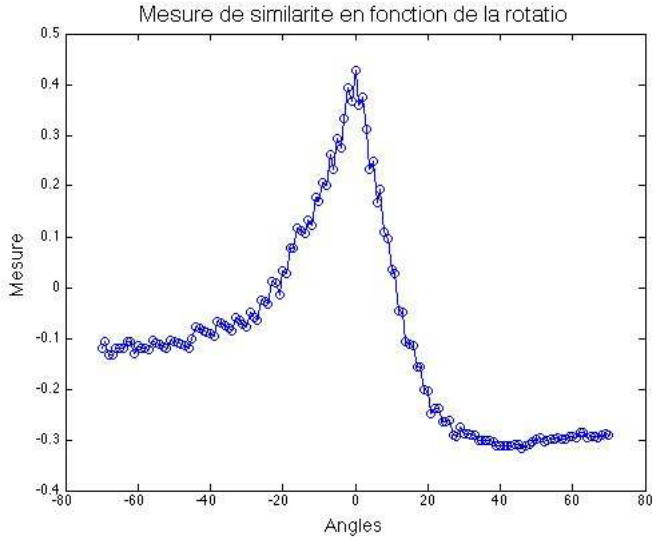


FIGURE 2.14 – Mesure de similarité (ρ de Spearman) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	338,64	111	215
Algorithme génétique	$6,3.10^3$	100	5050
Pattern Search	522,34	86	451

TABLE 2.13 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour le ρ de Spearman et le paysage d'Alsace.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	10,07	10,06	$[-0,09; 0,02; -0,38]$	$[0,02; 0,05; -0,53]$	$1,01$	$0,99$
Algorithme génétique		7,38	$[2,79; -3,23; -62,36]$	$[2,83; -3,26; -61,96]$	0,72	0,72
Pattern Search		7,41	$[0; 0,35; -57,79]$	$[0; 0,35; -57,79]$	1	1

TABLE 2.14 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour le ρ de Spearman et un paysage d'Alsace.

Nous n'allons pas retenir cette méthode étant donné les mauvaises valeurs obtenues (aux tableaux 2.13 et 2.14), notamment car la valeur du $PSNR$ diminue ainsi que le temps de calculs qui est largement plus élevé que dans les autres mesures considérées.

2.2.4 Mesure de similarité : SSIM

L'allure ne change pas à la figure 2.15 ; les commentaires restent similaires aux précédents.

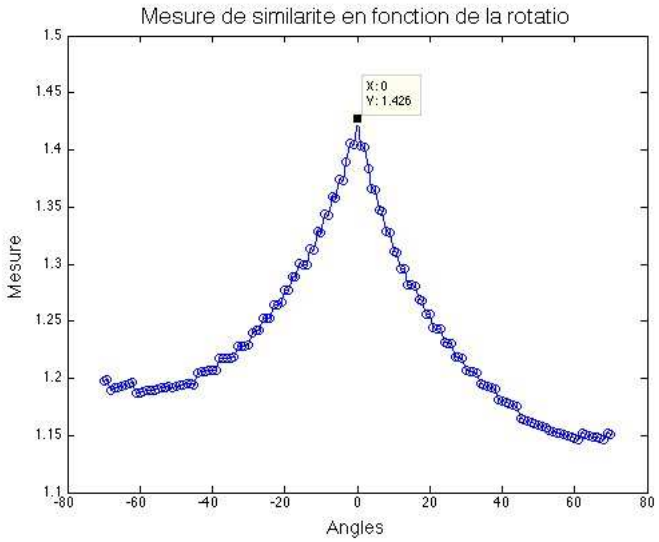
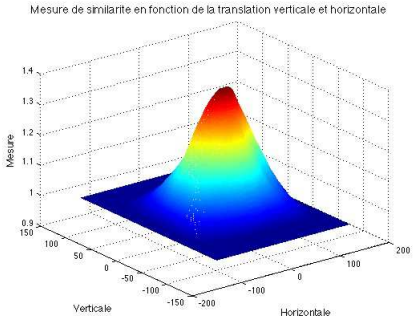
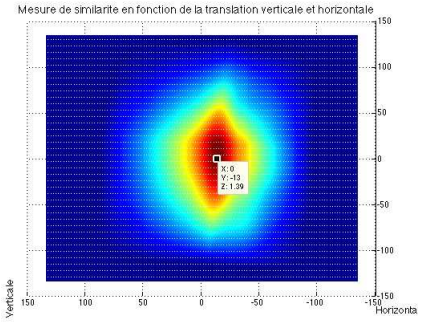


FIGURE 2.15 – Mesure de similarité (SSIM) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.

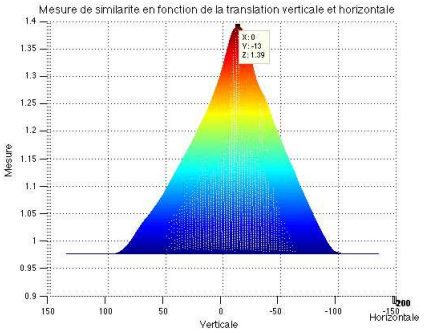
Le maximum n'est pas celui trouvé par la méthode et il est clair que le recalage ne sera pas réussi. Concernant les tableaux de résultats 2.15 et 2.16, l'erreur n'est pas aussi basse que nous l'espérons et les valeurs du $PSNR$ n'ont pas énormément augmenté voire par du tout dans le cas de la méthode de Nelder-Mead.



(a) Mesure en fonction des déplacements verticaux et horizontaux h et k respectivement



(b) Vue du dessus



(c) Vue latérale

FIGURE 2.16 – Mesure de similarité (SSIM) en fonction des paramètres de translation.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	8,28	122	603
Algorithme génétique	57,04	86	4350
Pattern Search	2.33	22	129

TABLE 2.15 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour le SSIM et le paysage d'Alsace.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	10,07	10,07	$[0; 0, 26.10^{-3}; 0]$	$[0; 0, 26.10^{-3}; 0]$	1	1
Algorithme génétique		10,09	$[-0, 49; 20, 98; -1, 94]$	$[-0, 43; 20, 99; -2]$	$1,05$	$1,04$
Pattern Search		10.09	$[0; 0, 5; 0]$	$[0; 0, 5; 0]$	1	1

TABLE 2.16 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour le SSIM et un paysage d'Alsace.

2.3 Première discussion

Au vu des différentes analyses qui ont été réalisées sur ces « simples » exemples, nous pouvons déduire que les méthodes employées dans le cas d'une image d'un paysage ne sont pas bien adaptées. Par contre, nous pouvons retenir les combinaisons : coefficient de Pearson avec la méthode de Pattern Search et la mesure SSIM avec l'Algorithme Génétique dans le cas d'images du même types que le logo (planétaires par exemple).

Dans le but d'appuyer ces observations, appliquons ces algorithmes à des images un peu plus concrètes : la Lune et la Nébuleuse d'Orion (présentées dans les figures 2.17 à la page 129 et 2.22 à la page 135 respectivement). Afin de vérifier que l'analyse s'applique bien sur des images concrètes et non pas sur un logo par exemple, l'image de détection sera identique à l'image de rotation mais également tournée de 10 degrés comme nous l'avons fait pour les deux premières images.

Par contre, nous ne retiendrons pas par la suite les mesures de similarité : le Minimum Ratio et le ρ de Spearman. En effet, la première ne nous donne pas des fonctions bien définies de sorte à trouver un maximum ; avec cette mesure, le problème est mal posé. Concernant la deuxième, la complexité plus élevée se ressent fortement lorsque nous désirons avoir des résultats : le temps d'exécution est beaucoup plus élevé. En plus de cela, nous n'avons pas obtenu de bons paramètres de recalage pour aucune des combinaisons contenant ces mesures.

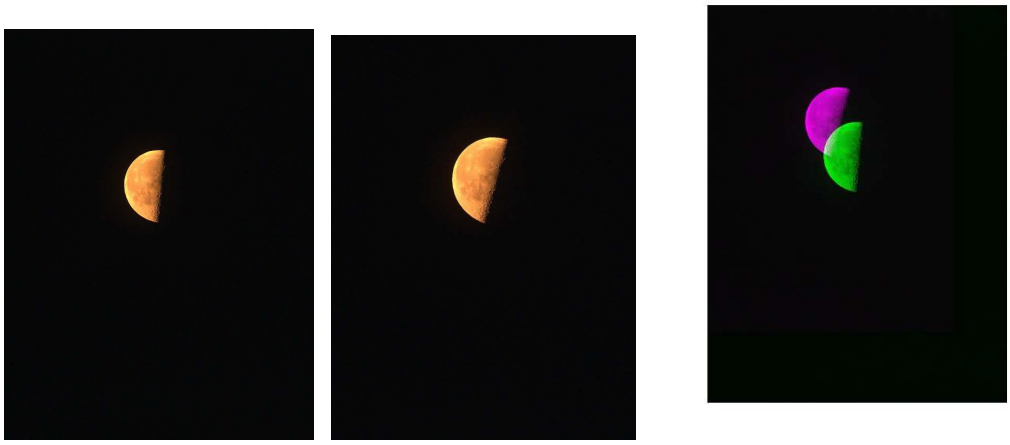
Dans la suite de cette analyse, nous allons garder les mesures du coefficient de Pearson et SSIM. Elles donnent toutes deux au moins une fois un bon résultat en un temps raisonnable ; alors que les exemples pris, nous les pensions « simples » et académiques.

2.4 La Lune

L'image de la Lune ainsi qu'une deuxième tournée de 10 degrés vers la droite sont représentées à la figure 2.17 accompagnées de la superpositions des deux illustrant leurs différences.

2.4.1 Mesure de similarité : Coefficient de Pearson

Les figures 2.18 et 2.19 montrent la manière dont varie la mesure de similarité en fonction de l'angle θ dans un premier temps et ensuite de la translation verticale k et horizontale h. Voyons comment se comportent les méthodes d'optimisation.



(a) Image de référence (b) Image de détection (c) Superposition des images sans recalage

FIGURE 2.17 – La Lune (1024×773).

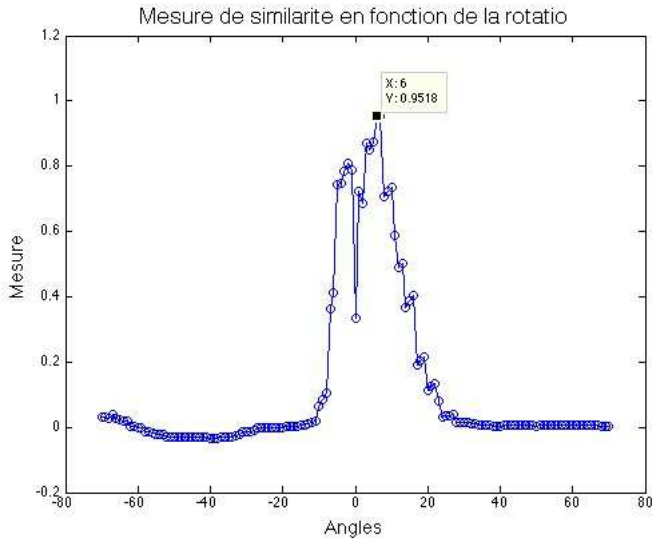
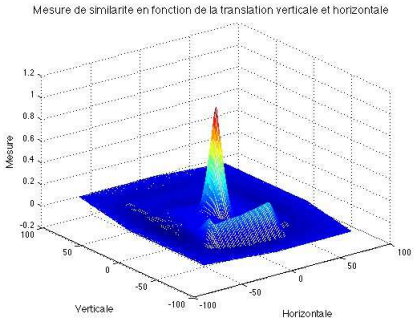
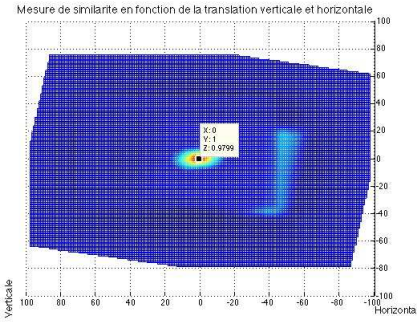


FIGURE 2.18 – Mesure de similarité (coefficient de Pearson) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.

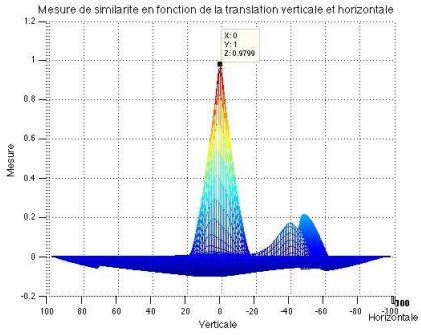
Les résultats des différentes combinaisons, en considérant le coefficient de Pearson comme mesure, sont indiqués dans les tableaux 2.17 et 2.18.



(a) Mesure en fonction des déplacements verticaux et horizontaux k et h respectivement.



(b) Vue du dessus



(c) Vue latérale

FIGURE 2.19 – Mesure de similarité (coefficient de Pearson) en fonction des paramètres de translation.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	5,59	248	456
Algorithme génétique	58,21	100	5050
Pattern Search	4,06	74	338

TABLE 2.17 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour le coefficient de Pearson et la Lune.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	19,26	31,76	$[-0,57; 3,95; 8,29]$	$[0,58; 4,4023; 8,15]$	$1,06$	$0,94$
Algorithme génétique		32,20	$[9,23; -0,57; 1,13]$	$[10,65; -0,09; 0,9]$	$0,08$	$0,07$
Pattern Search		32,89	$[10,65; -0,06; 1]$	$[10,65; -0,06; 1]$	$0,07$	$0,07$

TABLE 2.18 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour le coefficient de Pearson et la Lune.

Les résultats sont meilleurs que dans les exemples précédents : les algorithmes sont tous plutôt bons lorsque nous considérons le coefficient de Pearson comme mesure de similarité. Lorsque nous regardons de plus près, l'Algorithme Génétique et Pattern Search possèdent une meilleure précision quant à l'estimation de l'angle de rotation. De plus, ce sont ces deux derniers qui augmentent le plus la valeur du $PSNR$, bien que la différence avec celle obtenue avec la méthode de Nelder-Mead ne soit pas si différente. Concernant le temps d'exécution, la méthode de Pattern Search indique un temps inférieur. Le constat est similaire à propos du nombre d'évaluation de la fonction. L'Algorithme Génétique est beaucoup plus couteux que les deux autres tant en temps d'exécution qu'en nombre d'évaluations de la fonction.

2.4.2 Mesure de similarité : SSIM

Les figures 2.20 et 2.21 donnent une représentation des allures de la mesure en fonction des paramètres de la transformation géométrique.

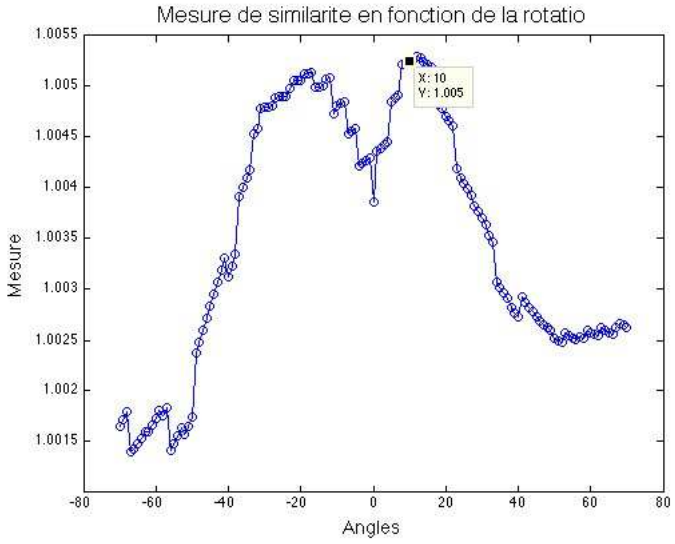
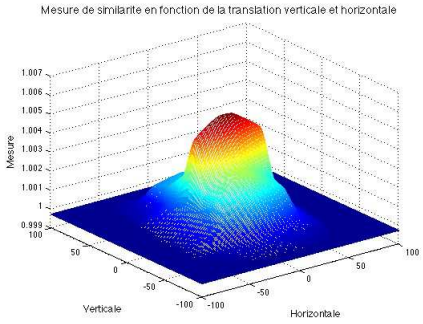
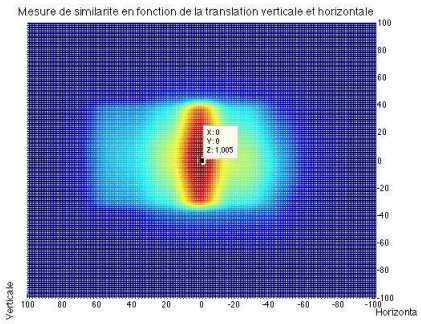


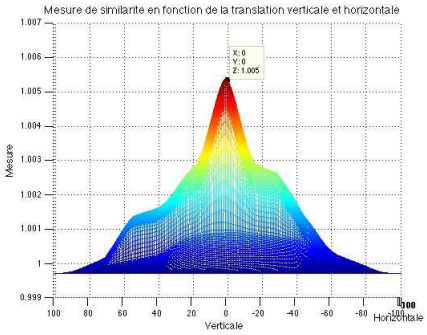
FIGURE 2.20 – Mesure de similarité (SSIM) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.



(a) Mesure en fonction des déplacements verticaux et horizontaux h et k respectivement



(b) Vue du dessus



(c) Vue latérale

FIGURE 2.21 – Mesure de similarité (SSIM) en fonction des paramètres de translation.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	2,05	67	127
Algorithme génétique	38,27	51	2600
Pattern Search	5,86	86	409

TABLE 2.19 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour SSIM et la Lune.

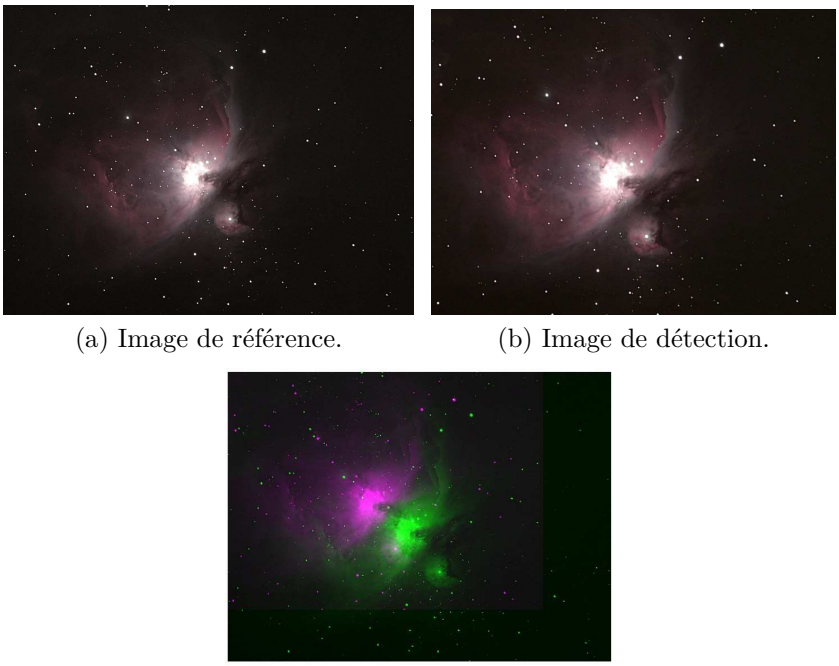
Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	19,26	22,54	$[3,05; -0,95; 6]$	$[3,88; -0,51; 6]$	$0,69$	$0,61$
Algorithme génétique		24,34	$[-7,39; -2,37; 4,82]$	$[-7,55; -2,49; 5]$	$1,74$	$1,76$
Pattern Search		30,06	$[10,36; -0.49; 1]$	$[10,36; -0,49; 1]$	$0,04$	$0,04$

TABLE 2.20 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour SSIM et la Lune.

Encore une fois, Pattern Search se montre plus performant que les autres mesures.

2.5 Nébuleuse d'Orion

À présent, considérons un objet du ciel profond qui est la Nébuleuse d'Orion à la figure 2.22. Celle-ci a également été tournée de 10 degrés vers la droite pour obtenir l'image de détection.



(a) Image de référence.

(b) Image de détection.

(c) Superposition des images sans recalage.

FIGURE 2.22 – Nébuleuse d'Orion (773×1024).

2.5.1 Mesure de similarité : Coefficient de Pearson

Les figures 2.23 et 2.24 illustrent la mesure de similarité en fonction des paramètres (angle de rotation, déplacements verticaux et horizontaux).

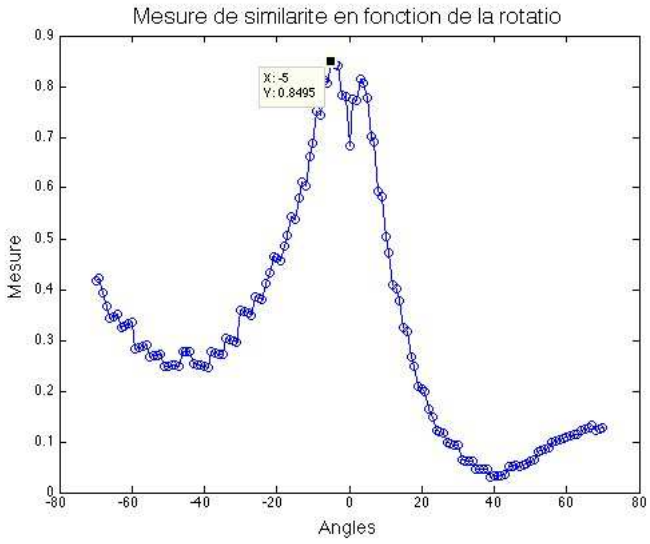
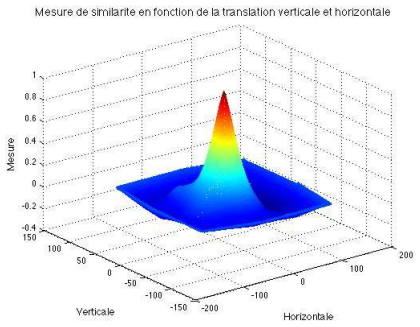
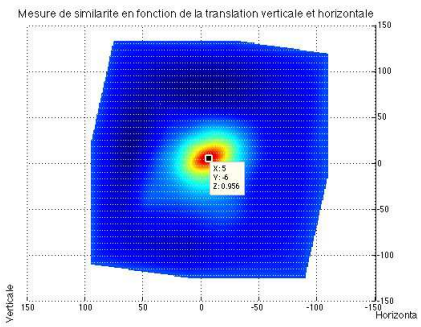


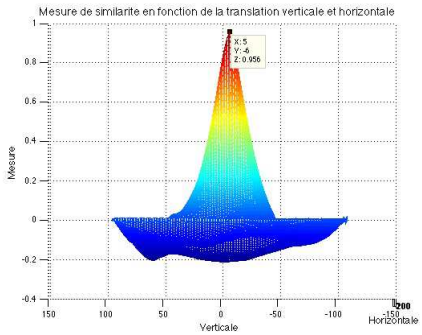
FIGURE 2.23 – Mesure de similarité (coefficient de Pearson) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.



(a) Mesure en fonction des déplacements verticaux et horizontaux h et k respectivement.



(b) Vue du dessus.



(c) Vue latérale.

FIGURE 2.24 – Mesure de similarité (coefficient de Pearson) en fonction des paramètres de translation.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	5,02	191	351
Algorithme génétique	123,35	192	9650
Pattern Search	7,05	108	521

TABLE 2.21 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour le coefficient de Pearson et la Nébuleuse d'Orion.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	18,01	24,21	$[4,02; 8,66; 4,27]$	$[4,028,724,18]$	$0,59$	$0,59$
Algorithme génétique		24,79	$[9,49; 4,89; -3,62]$	$[9,79; 5,08; -3,76]$	$0,05$	$0,02$
Pattern Search		24,48	$[7,65; 7,26; -1,29]$	$[7,65; 7,26; -1,29]$	$0,24$	$0,24$

TABLE 2.22 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour le coefficient de Pearson et la Nébuleuse d'Orion.

Nous pouvons constater que l'Algorithme Génétique donne une plus petite erreur de la valeur de θ que les deux autres algorithmes. Cependant, cela est au prix d'un temps d'exécution de 123,35 s contre 5,02 s pour la méthode d'optimisation de Nelder-Mead. Dans ce cas-ci, les trois méthodes donnent d'assez bons résultats numériques comparés à d'autres que nous avons vus précédemment.

2.5.2 Mesure de similarité : SSIM

La figure 2.25 montre que la fonction possède deux maxima locaux pour le paramètre de rotation. Par contre, à la figure 2.26, nous voyons distinctement la présence du maximum global.

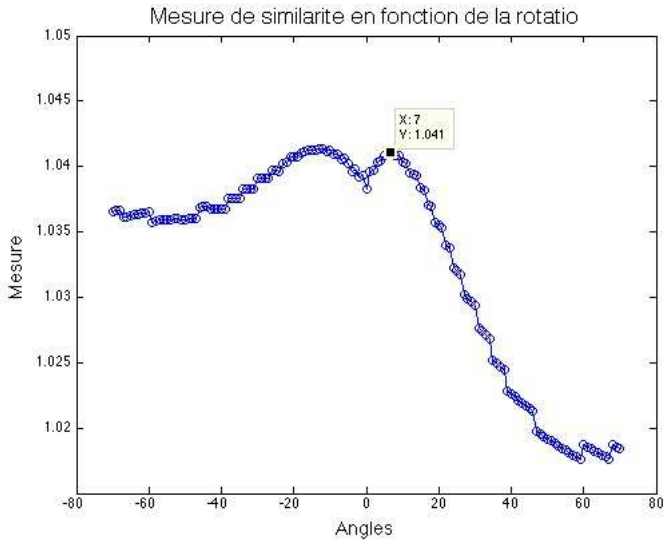
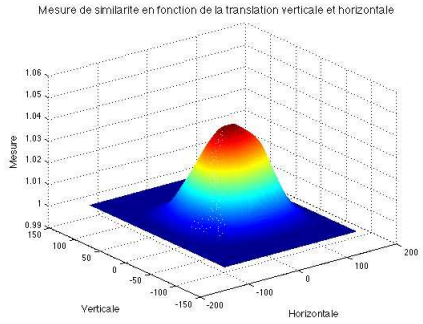
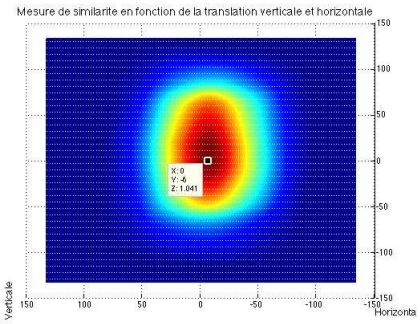


FIGURE 2.25 – Mesure de similarité (SSIM) en fonction de l'angle de rotation θ pour les paramètres de translation h et k fixés à 0 tous deux.

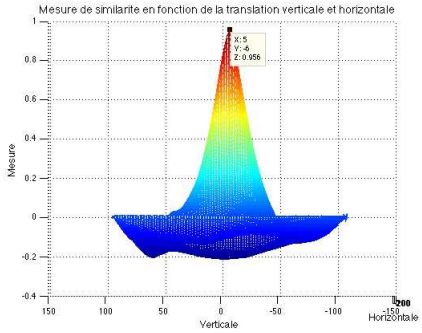
Nous pouvons constater que, dans les tableaux 2.23 et 2.24, les algorithmes ne donnent pas les bons paramètres de recalage. Par exemple, bien que la valeur du $PSNR$ soit la plus élevée pour l'Algorithme Génétique, il a trouvé le deuxième maximum de la fonction présentée à la figure 2.25 et pas celui qui se trouve sur la droite de l'image aux environs de 7 degrés en abscisses.



(a) Mesure en fonction des déplacements verticaux et horizontaux h et k respectivement.



(b) Vue du dessus.



(c) Vue latérale.

FIGURE 2.26 – Mesure de similarité (SSIM) en fonction des paramètres de translation.

Méthodes d'optimisation	Temps d'exécution	Nombre d'itérations (ou générations)	Nombre d'évaluations de la fonction
Nelder-Mead	2,13	69	136
Algorithme génétique	34,69	51	2600
Pattern Search	5,61	74	378

TABLE 2.23 – Résultats du temps d'exécution, nombre d'itérations et nombre d'évaluations des méthodes d'optimisation pour SSIM et la Nébuleuse d'Orion.

Méthodes d'optimisation	$PSNR$		Paramètres $[\theta; h; k]$		Erreur relative sur θ	
	Avant recalage	Après recalage	1^e	2^e	1^e	2^e
Nelder-Mead	18,01	18,88	$[2,67-14]$	$[2,38;-1,66;3,99]$	$0,73$	$0,76$
Algorithme génétique		22,99	$[-9,76;6,42;2,04]$	$[-9,79;6,52;2]$	$1,98$	$1,98$
Pattern Search		19,46	$[4,92;-1,57;1]$	$[4,92;-1,57;1]$	$0,51$	$0,51$

TABLE 2.24 – Résultats du $PSNR$, des paramètres et de l'erreur relative obtenus par les méthodes d'optimisation pour SSIM et la Nébuleuse d'Orion.

Images	Combinaisons	Temps d'exécution	Nombre d'évaluations de la fonction	$PSNR$ final	Erreur relative
Lune	Pearson - Génétique	58,21	5050	32,20	0,07
	Pearson - Pattern S.	4,06	338	32,89	0,07
	SSIM - Pattern S.	5,86	409	30,06	0,04
Nébuleuse d'Orion	Pearson - Génétique	123,35	9650	24,79	0,02
	Pearson - Pattern S.	7,05	521	24,48	0,24
	SSIM - Pattern S.	5,61	378	19,46	0,51

TABLE 2.25 – Récapitulatif des combinaisons retenues lors de la deuxième analyse.

2.6 Seconde discussion

Il est clair que les combinaisons qui ressortent de cette dernière analyse sont :

- Le coefficient de Pearson avec l'Algorithme Génétique ;
- Le coefficient de Pearson avec l'algorithme Pattern Search ;
- Le SSIM avec l'algorithme Pattern Search.

Donnons un tableau récapitulatif (tableau 2.25) contenant les temps d'exécution, les nombres d'évaluation de la fonction, les $PSNR$ après recalage et les erreurs relatives sur le paramètre de rotation pour chacune de ces combinaisons relevées.

Ce tableau permet d'avoir une meilleure visualisation de la situation. Par exemple, lorsque nous regardons l'erreur relative de chaque combinaison, nous pouvons constater que le coefficient de Pearson mêlé avec l'Algorithme Génétique donne une erreur plus petite dans le cas des objets du ciel profond. Par contre, SSIM combiné avec la méthode Pattern Search est un mauvais candidat comparé aux autres erreurs même concernant la valeur du $PSNR$.

Ensuite, le nombre d'évaluations de la fonction est moindre lorsque nous considérons la méthode de Pattern Search, en particulier lorsqu'il est combiné avec le coefficient de Pearson pour les deux images. Remarquons que nous omettons un commentaire sur la combinaison de SSIM avec Pattern Search étant donné qu'il s'agit de celle qui donne les moins bons résultats.

Dans la section suivante, nous devrons appliquer notre choix de combinaisons à des objets concrets dont nous ne connaissons pas les paramètres de recalage. Par conséquent, il faut que nous fassions un choix et ce, de manière raisonnable et réfléchie. Un exemple de critère est l'erreur relative et la valeur du $PSNR$. Dans ce cas-ci, les combinaisons de SSIM - Pattern Search et Pearson - Pattern Search sont de bons candidats concernant les images planétaires. Par rapport aux images du ciel profond, la combinaison

Pearson - Génétique est le seul candidat intéressant, plus particulièrement en analysant l'erreur relative.

La figure 2.27 donne le résultat des différentes combinaisons retenues. Visuellement, les résultats sont assez satisfaisants. De plus, nous pouvons mieux choisir une des deux combinaisons retenues pour l'image de la Lune : l'image recalée par le coefficient de Pearson avec l'algorithme du Pattern Search donne un meilleur résultat visuel.

(a) Lune recalée avec Pearson - Pattern Search

(b) Lune recalée avec SSIM - Pattern Search

(c) Nébuleuse d'Orion recalée avec Pearson - Génétique

FIGURE 2.27 – Résultats des différents recalages.

Chapitre 3

Applications à des exemples concrets

Maintenant que nous avons analysé les différentes combinaisons possibles de mesures de similarité avec les méthodes d'optimisation, nous nous intéressons à des images dont nous ne connaissons pas les paramètres de recalage.

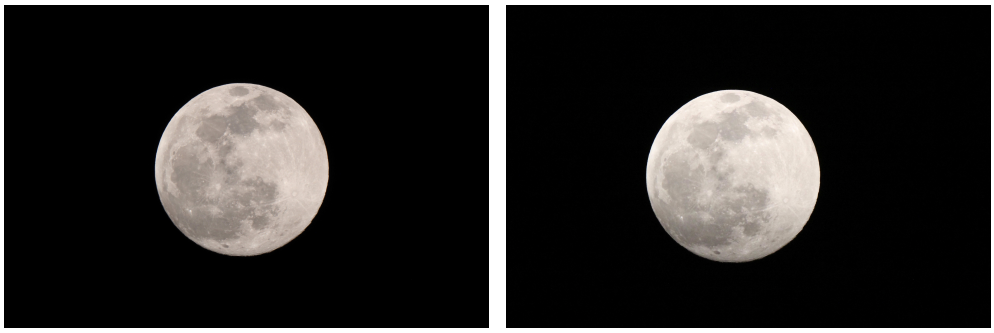
3.1 Pleine Lune

L'image 3.1 donne la superposition de deux images de la pleine Lune qui ont été prises durant la même nuit mais à des instants différents. Si nous regardons les deux images prises séparément, nous pouvons voir une légère différence de contraste ; cela est dû au temps de pose ainsi du fait que les images n'ont pas été prises au même instant.

Lorsque nous appliquons la combinaison de la mesure de similarité du coefficient de Pearson avec la méthode de Pattern Search, nous obtenons le résultat à la figure 3.2. La valeur du $PSNR$ au départ est de 17,67 et devient 29,23 après recalage. Il y a donc une belle augmentation. De plus, les paramètres de recalage trouvés par l'algorithme sont

$$[0; 3,27; -1,96].$$

Nous pouvons remarquer qu'il n'y a pas eu de rotation mais uniquement des déplacements verticaux et horizontaux entre les deux photographies.



(a) Image de référence (b) Image de détection

(c) Superposition des images sans recalage

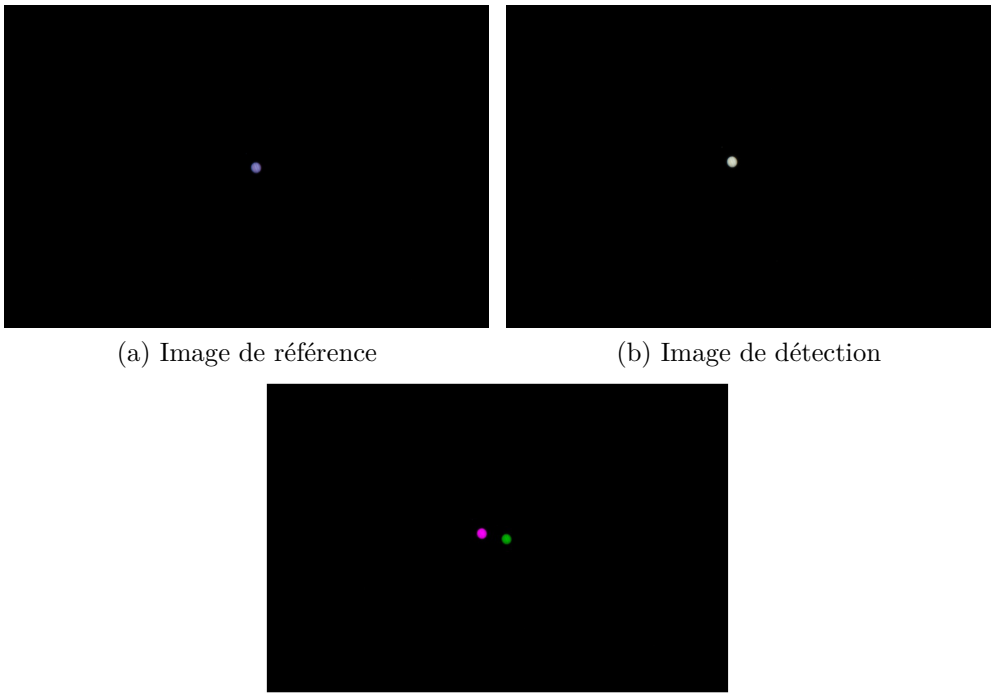
FIGURE 3.1 – Pleine Lune (3456×5184).

FIGURE 3.2 – Résultat du recalage de la pleine Lune.

3.2 Jupiter

Nous pouvons constater le déplacement de la planète durant les différents instants que les images ont été prises ainsi que la différence de luminosité.

Le résultat du recalage est plutôt bon, même pour un petit objet avec des



(a) Image de référence (b) Image de détection

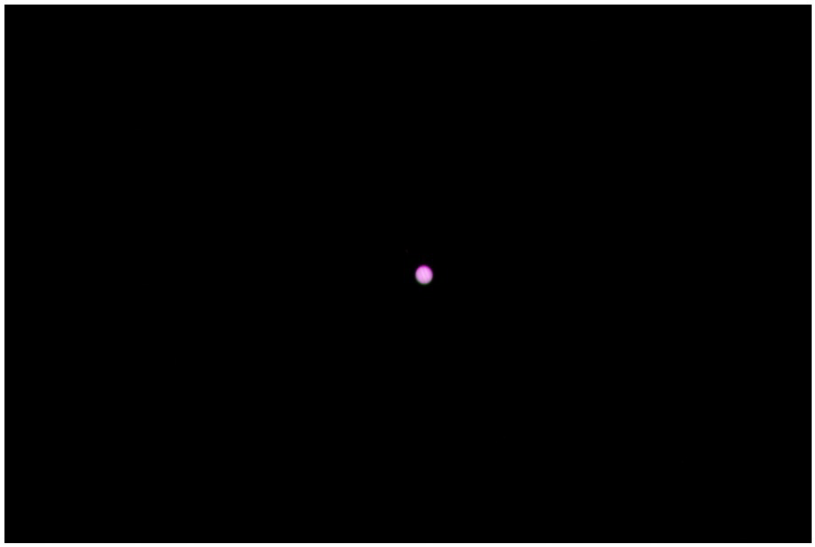
(c) Superposition des images sans recalage

FIGURE 3.3 – Jupiter (3456×5184).

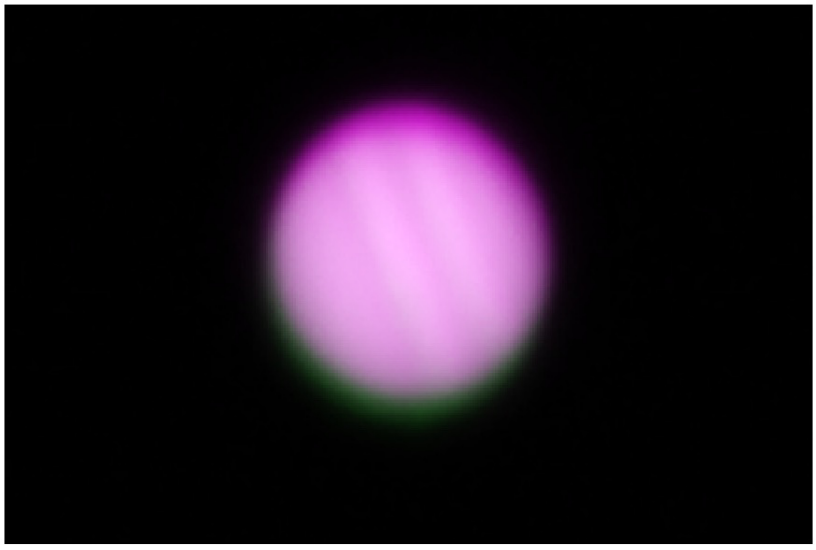
constrates différents. Le fait que nous n'ayons pas une image possédant uniquement des nuances de gris vient du fait que la luminosité des deux images est distincte dès le départ. Concernant la valeur du $PSNR$ au départ, elle est à 33,39 et devient après recalage 43,05. Les paramètres de recalage obtenus sont

$$[1,78; 6,58; -0,91].$$

Il y a eu une légère rotation. La figure 3.4 montre le résultat ainsi qu'un zoom pour avoir un aperçu de la qualité visuelle.



(a) Images recalées



(b) Zoom

FIGURE 3.4 – Résultat du recalage de la planète Jupiter.

Conclusion et pistes d'amélioration

Tout au long de ce mémoire, nous nous sommes initiés au traitement d'images numériques : comment nous devons manipuler les images ; les différents outils d'amélioration de la qualité (déconvolution par exemple) et surtout, le recalage d'images. Ce dernier se montre très utile lorsque nous souhaitons simuler une longue pose par une suite de courtes poses. En effet, nous cherchons à obtenir davantage d'informations grâce à une série de photographies d'un même objet. Plus la pose est longue, plus les photosites capturent des photons et donc des électrons qui sont convertis en pixels [7].

Ensuite, nous avons analysé le problème que nous allions devoir résoudre pour le recalage d'images : nous avons choisi d'optimiser une mesure de similarité. Cette dernière phrase implique deux termes importants : optimisation et mesure de similarité. Il existe une série de méthodes d'optimisation et de mesures de (di-) similarité. Nous avons choisi de maximiser une mesure de similarité qui plus est non différentiable ; cela restreint le champ des possibilités pour les considérations de méthodes d'optimisation. En effet, trois algorithmes nous ont intéressés : Nelder-Mead, Algorithme Génétique et Pattern Search. Leur point commun est que ce sont toutes des méthodes de recherche directes : elles ne requièrent pas le calcul du gradient et cela leur en donne un bon avantage pour notre problème.

Certains résultats obtenus lors du recalage sont efficaces pour les images planétaires. Nous avons pu retenir que l'algorithme de Pattern Search convient bien pour ce type de problème, plus particulièrement lorsqu'il est combiné avec la mesure de similarité du coefficient de Pearson. Concernant les images du ciel profond, nous devons continuer à trouver un algorithme plus convaincant car, même s'il a donné un bon résultat pour la Nébuleuse d'Orion, l'algorithme génétique ne fonctionne pas à tous les coups pour une raison encore obscure (cela est probablement dû à la génération aléatoire de la population initiale). Une première manière d'améliorer ce problème serait d'exploiter davantage les différentes options que propose la fonction MATLAB.

D'ailleurs, dans ce mémoire, nous n'avons pas exploité toutes les options proposées par MATLAB par rapport aux méthodes d'optimisation. Par exemple, s'il y a une différence visible à l'oeil nu, nous pourrions diminuer la limite de la longueur de pas de la grille de valeurs pour la méthode de Pattern Search ; ou bien, augmenter le nombre maximum d'évaluations de la fonction.

Une autre idée d'amélioration des algorithmes est l'application de la méthode de multirésolution que nous avons brièvement abordé à la section 2. En effet, nous avons bien diminué la résolution des images une seule fois mais nous pourrions appliquer cette méthode de manière itérée.

D'autre part, nous n'avons considéré que des mesures de similarité. Nous aurions pu utiliser des mesures de dissimilarité où le problème d'optimisation revient à minimiser la fonction.

Pour terminer, le mémoire conduit à chercher une meilleure façon de recaler des images du ciel profond. Par exemple, nous avons abordé la transformée de Hough pour les cercles (section 1.4). En effet, cela est dans le but de repérer des cercles dans les images ; c'est à dire les étoiles pour les considérer comme points de détection. De cette manière, nous recalons les images à partir de ces cercles et non plus à partir de l'image entière (à résolution réduite) ; cela réduirait notamment le temps de calcul. Nous espérons que le résultat sera satisfaisant pour recaler des images d'objets du ciel profond ; mais cela est encore à tester et à analyser.

Bibliographie

[1] Ardeshir Goshtasby, A., 2012, *Image Regitration : Principles, Tools and Methods*, Springer, London

[2] Audet, Charles, and DennisJ. E. Jr., 2003, *Analysis of Generalized Pattern Searches* SIAM Journal on Optimization. Volume 13, Number 3, pp. 889–903

[3] Bodenhofer U., 2001-2002, *Genetic Algorithm : Theory and Applications (Lecture Notes, second edition)*, Johannes Kepler universität, Linz, Austria

[4] Chinneck J.W., 2014, *Practical Optimization : a Gentle Introduction*, http ://www.sce.carleton.ca/faculty/chinneck/po.html, consulté le 20 avril 2017

[5] Gonzalez, R. C., et al., 2004, *Digital Image Processing Using MATLAB*, Mc Graw Hill Education

[6] Graf B., Espic C., *Transformée de Hough*, http ://elynxsdk.free.fr/ext-docs/Demosaicing/more/news1/article-graf-espic.pdf, consulté le 12 avril 2017

[7] Legault, T., 2013, *Astrophotographie*, Eyrolles, 2^e édition, Paris

[8] Lobo F., et al., 2000, *Time complexity of genetic algorithms on exponentially scaled problems*, Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation, pp. 151-158

[9] Leontitsis A., *Spearman Rank Correlation*, https ://nl.mathworks.com/matlabcentral/fileexchange/4374-spearman-rank-correlation, MathWorks, File Exchange, consulté le 30 avril 2017

[10] Mathews J.H., Fink Kurtis D.,2004, *Numerical Methods Using Matlab*, 4^{th} Edition, pp. 430 - 436

[11] MathWorks, *fminsearch*, https ://nl.mathworks.com/help/matlab/ref/fminsearch.html, consulté le 9 mai 2017

[12] MathWorks, *ga*, https ://nl.mathworks.com/help/gads/ga.html, consulté le 10 mai 2017

[13] MathWorks, *imfindcircle*, https ://nl.mathworks.com/help/images/ref/imfindcircles.html, consulté le 11 avril 2017

[14] MathWorks, *Image Registration*, http ://nl.mathworks.com/discovery/image-registration.html, 10 mai 2016

[15] MathWorks, *patternsearch*, https ://nl.mathworks.com/help/gads/patternsearch.html, consulté le 12 avril 2017

[16] MathWorks, *psnr*, https ://nl.mathworks.com/help/images/ref/psnr.html, consulté le 10 avril 2017

[17] MathWorks, *snr*, https ://nl.mathworks.com/help/signal/ref/snr.html, consulté le 10 avril 2017

[18] Nicolay D., 2012, *Modélisation et apprentissage des réseaux de neurones artificiels*, Carletti Timoteo, Université de Namur

[19] Rhody H., *Lecture 10 : Hough Circle Transform*, Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology, 11 october 2005

[20] Singer S., 1999, *Complexity Analysis of Nelder-Mead Search Iterations*, Dubrvnik, Croatia, pp. 185-196

[21] Starck, J.-L., Murtagh, F., 2006, *Astronomical Image and Data Analysis*, Springer

[22] Torczon V., 1993, *On the convergence of pattern search algorithms*, Tech. Report 93-10, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, 77251-1892

[23] Torczon V., 1997, *Pattern search for nonlinear optimisation*, SIAM J. Optimization

[24] Vallée T., Yilildizoglu M.,7 septembre 2001, *Présentation des algorithmes génétiques et de leurs applications en économie*

[25] Wang Z., Bovik A. C.,et al., april 2004, *Image Quality Assessment : From Error Visibility to Structural Similarity*, IEEE Transactions on Image Processing, vol. 13, no. 4

[26] Wikipédia, *Algorithme génétique*, https ://fr.wikipedia.org/wiki/Algorithme_génétique, consulté le 10 mai 2017

[27] Wikipédia, *Circle Hough Transform*, https ://en.wikipedia.org/wiki/Circle_Hough_Transfo consulté le 11 avril 2017

[28] Wikipédia, *Digital image processing*, https ://en.wikipedia.org/wiki/Digital_image_process 10 mai 2016

[29] Wikipédia, *Méthode de Nelder-Mead*, https ://fr.wikipedia.org/wiki/Méthode_de_Nelder-Mead, consulté le 9 mai 2017

[30] Wikipedia, *Image noise*, https ://en.wikipedia.org/wiki/Image_noise, 10 mai 2016

Fonction principale : Opti.m

```
%_____________________________________________________________________
%_____________________________________________________________________
% Nom fichier : Opti.m
% Realise par : Mathilde AVILES-ROMARIZ
% But : Fonction principale de recalage pour l'etude de cas
%_____________________________________________________________________
%_____________________________________________________________________

clear all;
close all;
clc;

%_____________________________________________________________________
% Choix du type d'image (1. logo, 2. paysage, 3. Lune, 4. ... Nebuleuse, 5. Lune naturelle, 6. Jupiter)
% pour ensuite, appliquer a l'astrophotographie. Il s'agira ... de l'image de
% reference qui restera fixee tout au long du processus. ... Pour le 6.,
% on ne connait pas la transformation geometrique exacte.
%_____________________________________________________________________

choix_IMG = 1;

if (choix_IMG == 1)

    fixed = im2double(rgb2gray(imread('UNamur.JPG')));

elseif (choix_IMG == 2)

    fixed = im2double(rgb2gray(imread('Alsace.JPG')));

elseif (choix_IMG == 3)

    fixed = im2double(rgb2gray(imread('Lune.JPG')));

elseif (choix_IMG == 4)

    fixed = im2double(rgb2gray(imread('Nebuleuse.JPG')));

elseif (choix_IMG == 5)

    fixed = im2double(rgb2gray(imread('IMG_1173.JPG')));

elseif (choix_IMG == 6)

    fixed = im2double(rgb2gray(imread('IMG_1776.JPG')));

end
```

```
47
48 moving = charger(fixed, choix_IMG);% On charge l'image qui ...
       va etre modifiee en fonction du type d'image choisie.
49
50 figure;
51 b = imshowpair(fixed, moving,'Scaling', 'joint'); % ...
       Affichage des images sans recalage
52
53 peaksnr = psnr(moving, fixed) % PSNR des originaux
54
55
56 fixed_degr = degrade(fixed); % Degradation des images pour ...
       accelerer le processus
57 moving_degr = degrade(moving);
58
59 x_0 = [0 , 0 , 0]; % Point initial
60
61 %_____________________________________________________________
62 % Choix de la mesure de similarite
63 %_____________________________________________________________
64
65 choix_MESURE = 1;
66
67 fun = similarite(fixed_degr, moving_degr, choix_MESURE);
68
69 %_____________________________________________________________
70 % Methodes d'optimisation
71 %_____________________________________________________________
72
73 choix_METH = 2;
74
75 tic % Mesure du temps d'execution de la methode d'optimisation
76
77 [param_Opti1 iter] = fctOpti(fun, fixed_degr, moving_degr, ...
       x_0, choix_METH)
78
79 tps_exe = toc
80
81 param_Opti2 = fminsearch(@(param)-fun(fixed_degr, ...
       moving_degr, param),param_Opti1) % Affine la solution ...
       trouvee
82
83 %_____________________________________________________________
84 % Application des nouveaux parametres aux images originales
85 %_____________________________________________________________
86 [m n] = size(fixed);
87 m_r = m/100; % ratio de proportionalite
88
89 recal = geometricT([param_Opti2(1) param_Opti2(2)*m_r ...
       param_Opti2(3)*m_r], moving);
90
91 recal = redim(recal, size(fixed)); % enlever les bords en ...
       trop de l'image
```

```
92
93  peaksnrR = psnr(recal, fixed) % signal sur bruit apres recalage
94
95  figure;
96  a = imshowpair(fixed, recal,'Scaling', 'joint'); % ...
        affichage du resultat
97
98  %_____________________________________________________________________
99  % Analyse de l'erreur sur l'angle (avec la connaissance de ...
        la rotation effectuee)
100 %_____________________________________________________________________
101
102 err_theta1 = abs(param_Opti1(1) - 10) / 10
103 err_theta2 = abs(param_Opti2(1) - 10) / 10
104
105 %_____________________________________________________________________
106 % Representations graphiques
107 %_____________________________________________________________________
108
109 % Variation de l'angle et les deplacements verticaux et ...
        horizontaux fixes
110 theta_max = 70;
111 theta = [-theta_max : 1 : theta_max];
112 n = length(theta);
113 for i = 1 : n
114     Y(i) = fun(fixed_degr, moving_degr, [theta(i) ...
            param_Opti2(1) param_Opti2(2)]);
115 end
116
117 figure;
118 plot(theta, Y,'o-');
119 title('Mesure de similarite en fonction de la ...
        rotation','FontSize',16);
120 xlabel('Angles','FontSize',14);
121 ylabel('Mesure','FontSize',14);
122
123 % Faire varier la translation et fixer la rotation
124 [mx nx] = size(fixed_degr);
125 L = max(mx, nx);
126 [H, K] = meshgrid(-L : 1 : L);
127 for i = 1 : 2*L+1
128     for j = 1 : 2*L+1
129         T(i,j) = 10;
130         Z(i,j) = fun(fixed_degr,moving_degr, [T(i,j), ...
                H(i,j), K(i,j)]);
131     end
132 end
133
134 figure;
135 mesh(H,K,Z)
136 title('Mesure de similarite en fonction de la translation ...
        verticale et horizontale', 'FontSize',14);
137 xlabel('Horizontale','FontSize',12);
```

```
138 ylabel('Verticale','FontSize',12);
139 zlabel('Mesure','FontSize', 12);
```

fctOpti.m

```
1  %_____________________________________________________________________
2  %_____________________________________________________________________
3  % Nom fichier : fctOpti.m
4  % Realise par : Mathilde AVILES-ROMARIZ
5  % But : Fonction qui determine la methode d'optimisation a ...
       utiliser
6  % Input :
7  %           - fun : mesure de similarite ;
8  %           - X : image de reference ;
9  %           - Y : image a modifier ;
10 %           - x_0 : condition initiale ;
11 %           - choix : choix de la methode a utiliser.
12 % Output :
13 %           - param_Opti : parametres d'optimisation ...
       (angle, deplacements
14 %            horizontaux et verticaux) ;
15 %           - iter : les iterations que la methode a ...
       utilise pour sortir
16 %            la solution
17 %_____________________________________________________________________
18 %_____________________________________________________________________
19
20 function [param_Opti iter] = fctOpti(fun, X, Y, x_0, choix)
21
22     if (choix == 1)
23
24         [param_Opti fval exitval iter]= ...
               fminsearch(@(param)-fun(X,Y,param),x_0);
25
26     elseif (choix == 2)
27
28         [param_Opti fval exitval iter] = ...
               ga(@(param)-fun(X,Y,param), 3, [],[],[],[],[]);
29
30     elseif (choix == 3)
31
32         [param_Opti fval exitval iter] = ...
               patternsearch(@(param)-fun(X,Y,param), ...
               x_0,[],[],[],[],[]);
33
34     end
35
36 end
```

similarite.m

```
%_____________________________________________________________________
%_____________________________________________________________________
% Nom fichier : similarite.m
% Realise par : Mathilde AVILES-ROMARIZ
% But : Fonction qui determine la mesure de similarite choisie
% Input :
%           - X : image de reference ;
%           - Y : image de detection ;
%           - choix : choix de la mesure a utiliser.
% Output :
%           - mesure : mesure de similarite choisie.
%_____________________________________________________________________
%_____________________________________________________________________
function mesure = similarite(X, Y, choix)

    if (choix == 1)
        mesure = @pearson ;

    elseif (choix == 2)
        mesure = @minimumRatio ;

    elseif (choix == 3)
        mesure = @spearmanRho ;

    elseif (choix == 4)
        mesure = @ssimMoi ;

end
```

charger.m

```
%_____________________________________________________________________
%_____________________________________________________________________
% Nom fichier : charger.m
% Realise par : Mathilde AVILES-ROMARIZ
% But : Fonction permettant de charger les images ...
    correspondantes au choix
%       de l'utilisateur
% Input : - X l'image choisie dans le fichier Opti.m ;
%         - choix est le choix de l'utilisateur.
% Output : - Y l'image correspondante tournee de 10 degres.
%_____________________________________________________________________
%_____________________________________________________________________

```

```
14 function Y = charger(X, choix)
15
16     [m, n] = size(X); % Il faut que les dimensions soient ...
           identiques
17     Y = zeros([m,n]);
18
19     if (choix == 1)
20
21         innput = im2double(rgb2gray(imread('UNamur10.jpg')));
22
23     elseif(choix == 2)
24
25         innput = im2double(rgb2gray(imread('Alsace10.jpg')));
26
27     elseif (choix == 3)
28
29         innput = im2double(rgb2gray(imread('Lune10.jpg')));
30
31     elseif (choix == 4)
32
33         innput = ...
               im2double(rgb2gray(imread('Nebuleuse10.jpg')));
34
35     elseif (choix == 5)
36
37         innput = im2double(rgb2gray(imread('IMG_1220.JPG')));
38
39     elseif (choix == 6)
40
41         innput = im2double(rgb2gray(imread('IMG_1842.JPG')));
42
43     end
44
45     [o, p] = size(innput)
46     for i = 1 : o
47         for j = 1 : p
48             Y(i,j) = innput(i,j);
49         end
50     end
51
52 end
```

degrade.m

```
%_____________________________________________________________________
%_____________________________________________________________________
% Nom fichier : degrade.m
% Realise par : Mathilde AVILES-ROMARIZ
% But : Fonction permettant de reduire la resolution d'une ...
    image
% Input : - X a reduire
% Output : - Y reduite
%_____________________________________________________________________
%_____________________________________________________________________


% Fonction pour degrader une image (diminuer la resolution)
% On degrade d'office l'image pour qu'elle ait 100 lignes ...
    pour avoir le
% meme temps d'execution pour toutes les images bien que ...
    plus l'image sera
% grande, plus la degradation sera grande

function Y = degrade(X)
    [m n] = size(X);

    m_new = 100;
    n_new = (m_new/m)*n;

    Y = imresize(X,[m_new, n_new]); % la fonction met la ...
        dimension a l'unite superieure automatiquement
end
```

geometricT.m

```
%_____________________________________________________________________
%_____________________________________________________________________
% Nom fichier : geometricT.m
% Realise par : Mathilde AVILES-ROMARIZ
% But : Fonction de realiser la transformation geometrique ...
    d'un angle
% et d'une translation
% Input : - param est un vecteur qui contient les ...
    parametres de
%                 transformation de la forme [theta h k]
%         - Y est l'image a tourner
% Output : - Y_new est l'image modifiee par la transformation
%_____________________________________________________________________
%_____________________________________________________________________
```

```
13 function Y_new = geometricT(param, Y)
14
15     [m n] = size(Y);
16
17     theta = param(1);
18     h = param(2);
19     k = param(3);
20
21     Y_new = imrotate(Y, theta, 'bicubic');
22     Y_new = imtranslate(Y_new, [h , k]);
23
24 end
```

redim.m

```
1  %_____________________________________________________________________
2  %_____________________________________________________________________
3  % Nom fichier : redim.m
4  % Realise par : Mathilde AVILES-ROMARIZ
5  % But : Fonction qui permet de redimensionner une image en ...
       fonction des
6  % dimensions donnees en arguments
7  % Input : - X est l'image dont on doit modifier les dimensions
8  %         - dim les dimensions que doit avoir l'image ...
       resultante
9  % Output : - Y est l'image modifiee
10 %_____________________________________________________________________
11 %_____________________________________________________________________
12 function Y = redim(X,dim)
13     Y = zeros(dim);
14
15     for i = 1 : dim(1)
16         for j = 1 : dim(2)
17             Y(i,j) = Y(i,j) + X(i,j);
18         end;
19     end;
20 end
```

pearson.m

```
1  %_____________________________________________________________________
2  %_____________________________________________________________________
3  % Nom fichier : pearson.m
4  % Realise par : Mathilde AVILES-ROMARIZ
```

```
% But : Fonction permettant de calculer le coefficient de ... Pearson entre
%       deux images
% Input : - X l'image de reference ;
%         - Y l'image de detection.
%         - param application de la transformation ... geometrique ([theta, h , k]).
% Output : - r le coefficient de Pearson.
%_____________________________________________________________________
%_____________________________________________________________________


function r = pearson(X, Y, param)

Y = geometricT(param, Y);

[n m] = size(X); % n lignes et m colonnes
X_moy = 0;
Y_moy = 0;
sigma_X = 0;
sigma_Y = 0;

for i = 1:n
    for j = 1:m
        X_moy = X_moy + X(i,j);
        Y_moy = Y_moy + Y(i,j);
    end
end

X_moy = X_moy / (n*m);
Y_moy = Y_moy / (n*m);


for i = 1:n
    for j = 1:m
        sigma_X = sigma_X + ((X(i,j) - X_moy)^2) ;
        sigma_Y = sigma_Y + ((Y(i,j) - Y_moy)^2) ;
    end
end

sigma_X = sqrt((1/(n*m)) * sigma_X);
sigma_Y = sqrt((1/(n*m)) * sigma_Y);

r = 0;

for i = 1:n
    for j = 1:m
        r = r + (((X(i,j) - X_moy)/sigma_X)*((Y(i,j) - ... Y_moy)/sigma_Y));
    end
end

r = (1/(n*m))*r;
```

```
55 end
```

minimumRatio.m

```
1  %_____________________________________________________________________
2  %_____________________________________________________________________
3  % Nom fichier : minimumRatio.m
4  % Realise par : Mathilde AVILES-ROMARIZ
5  % But : Fonction permettant de calculer le Minimum Ratio entre
6  %       deux images
7  % Input : - X l'image de reference ;
8  %         - Y l'image de detection.
9  %         - param application de la transformation ...
       geometrique ([theta, h , k]).
10 % Output : - m_r le Minimum Ratio.
11 %_____________________________________________________________________
12 %_____________________________________________________________________
13
14 function m_r = minimumRatio(X,Y, param)
15
16     Y = geometricT(param, Y);
17
18     [m,n] = size(X);
19     r = zeros(m,n);
20     m_r = 0;
21
22     for i = 1:m
23         for j = 1:n
24             r(i,j) = min(X(i,j)/Y(i,j), Y(i,j)/X(i,j));
25             m_r = m_r + r(i,j);
26         end
27     end
28
29     m_r = m_r/(m*n);
30 end
```

spearmanRho.m

```
1  %_____________________________________________________________________
2  %_____________________________________________________________________
3  % Nom fichier : spearmanRho.m
4  % Realise par : Mathilde AVILES-ROMARIZ
5  % But : Fonction permettant de calculer le rho de Spearman ...
       entre
6  %       deux images
```

```
% Input : - X l'image de reference ;
%         - Y l'image de detection.
%         - param application de la transformation ...
    geometrique ([theta, h , k]).
% Output : - rho le rho de Spearman.
%_____________________________________________________________________
%_____________________________________________________________________

function rho = spearmanRho(X,Y, param)
    Y = geometricT(param, Y);

    [m,n] = size(X);
    Y = redim(Y,[m n]);
    [o,p] = size(Y);

    X = reshape(X',m*n,1);
    Y = reshape(Y',o*p,1);

    if(size(X,1) ~= size(Y,1))
        vide = zeros(size(X,1));
        for i = 1:size(X,1)
           vide(i) = vide(i) + Y(i);
        end
        Y = vide;
    end

    if size(X,1)~=size(Y,1)
        error('x and y must have equal number of rows.');
    end

    % Find the data length
    N = length(X);
    % Get the ranks of x
    R = crank(X)';

    for i=1:size(Y,2)
        % Get the ranks of y
        S = crank(Y(:,i))';
        % Calculate the correlation coefficient
        rho(i) = 1-6*sum((R-S).^2)/N/(N^2-1);
    end
end
```

ssimMoi.m

```
%_____________________________________________________________________
%_____________________________________________________________________
% Nom fichier : ssimMoi.m
% Realise par : Mathilde AVILES-ROMARIZ
```

```
5  % But : Fonction permettant de calculer le SSIM (Structural ...
       SIMilarity)
6  %       entre deux images
7  % Input : - X l'image de reference ;
8  %          - Y l'image de detection.
9  %          - param application de la transformation ...
       geometrique ([theta, h , k]).
10 % Output : - s le SSIM.
11 %_____________________________________________________________
12 %_____________________________________________________________
13
14 function s = ssimMoi(X,Y, param)
15
16 Y = geometricT(param, Y);
17 [n m] = size(X); % n lignes et m colonnes
18 Y = redim(Y,[n m]);
19 k_1 = 0.01;
20 k_2 = 0.03;
21 L = 255;
22
23 c_1 = (k_1*L)^2;
24 c_2 = (k_2*L)^2;
25 c_3 = c_2/2;
26
27 X_moy = 0;
28 Y_moy = 0;
29 sigma_X = 0;
30 sigma_Y = 0;
31
32 for i = 1:n
33     for j = 1:m
34         X_moy = X_moy + X(i,j);
35         Y_moy = Y_moy + Y(i,j);
36     end
37 end
38
39 X_moy = X_moy / (n*m);
40 Y_moy = Y_moy / (n*m);
41
42
43 for i = 1:n
44     for j = 1:m
45         sigma_X = sigma_X + ((X(i,j) - X_moy)^2) ;
46         sigma_Y = sigma_Y + ((Y(i,j) - Y_moy)^2) ;
47     end
48 end
49
50 sigma_X = sqrt((1/(n*m)) * sigma_X);
51 sigma_Y = sqrt((1/(n*m)) * sigma_Y);
52
53 XY = X.*Y;
54 [o p] = size(XY);
55 XY_moy = 0;
```

```
for i = 1:o
    for j = 1:p
        XY_moy = XY_moy + XY(i,j);
    end
end

XY_moy = XY_moy/(o*p);
sigma_XY = XY_moy - (X_moy*Y_moy);

s = ((2*X_moy * Y_moy+c_1)*(2*sigma_XY+c_2))/ ((X_moy^2 + ...
   Y_moy^2+c_1)*(sigma_X^2 + sigma_Y^2 + c_2));
end
```