SIMES – 961620
Système d'Information Multimédia
Pour l'Environnement Subsaharien

DATA PRE-PROCESSING

Deliverable number : D7
Nature : P
Contractual Date of Delivery : 14 November 1998
Task WP1.3 : Data pre-processing

Nom du rédacteur : Paul S. SAWADOGO
Institut : ESI
Adresse : 01 BP1091 Bobo-Dioulasso 01
Burkina Faso
Spaul@esi.univ-ouga.bf

ABSTACT

This paper is the deliverable due at the end of the task WP1.3 of workpackage 1 of SIMES WISE-DEV project. It describe some algorithms of image preprocessing which can be used on remote sensing images on subsaharian environment. These algorithms are mainly based on image restoration and contrast enhancement technics.

Key Words : Image restoration, contrast enhancement, filtering.

RESUME

Ce document constitue le délivrable attendu à la fin de la tâche WP1.3 du workpackage 1 figurant dans le projet SIMES WISE-DEV. Il décrit quelques algorithmes de programmes de pré-traitement d'images en mesure d'être employés sur des images numériques de l'environnement subsaharien prises à partir de capteurs aéroportés ou embarqués sur des satellites. Ces algorithmes sont essentiellement ceux destinés à l'amélioration et la restauration des images.

Mots clés : amélioration d'images, restauration d'images, filtrage.

SOMMAIRE

INTRODUCTION

Les images acquises sont le plus souvent entachées de parasites ou bruits. Le traitement à leur appliquer dépend de l'information que l'on veut en extraire. En effet, le dispositif et les conditions d'acquisition n'étant pas toujours parfaits, l'image peut présenter des distorsions ou un flou ; pour s'en débarrasser, les techniques de restauration d'images peuvent être employées.

Il peut arriver que malgré la suppression du bruit, l'image soit encore de mauvaise qualité, rendant de ce fait l'interprétation difficile ; les techniques d'amélioration d'images vont nous permettre alors de donner une meilleure qualité à l'image en lui rendant un aspect visuel plus agréable.

Ce document est un inventaire non exhaustif des méthodes existant en amélioration et restauration d'images. Dans une première partie nous en présentons une description fonctionnelle, c'est-à-dire leurs fondements théoriques. Dans la deuxième partie, nous donnons les algorithmes réalisant quelques-unes d'entre elles avec les codes sources associés. Dans la troisième partie figurent les résultats des différents tests des programmes correspondants.

PREMIERE PARTIE

Description fonctionnelle des algorithmes de pré-traitement d'images centrés sur l'amélioration et la restauration d'images

Une image numérique peut être considérée comme une matrice de points appelés pixels repérés par leurs coordonnées *(x, y)*. A chacun d'entre eux est associée une valeur correspondant à la mesure d'un phénomène en ce lieu que l'on représente généralement par une fonction de deux variables $f : R \times R \rightarrow R$.

Pour améliorer le contraste d'une image, il est nécessaire d'avoir une idée de la dispersion des valeurs des pixels qui la composent. Pour cela, on peut la visualiser, c'est-à-dire l'afficher à l'écran ou l'imprimer de sorte à en avoir une appréciation visuelle subjective, ou bien dresser son histogramme qui en offrira une perception quantitative et objective. Si l'histogramme ne nous donne pas d'information spécifique sur ce que contient l'image, en revanche sa forme nous permet de savoir si l'image est faiblement ou fortement contrastée, ce qui peut aider au choix de la méthode appropriée pour son amélioration.

L'histogramme d'une image dont les valeurs des pixels sont dans l'intervalle [0, L-1], est une fonction discrète p définie par :

$p : [0, L-1] \rightarrow [0, 1]$

$$p(r_k) = \frac{n_k}{n} \qquad (1)$$

où r_k est la $k^{ième}$ valeur de pixel,

n_k le nombre total de pixels qui ont cette valeur,

$k = 0, 1, \ldots, L-1$,

n le nombre total de pixels.

NB : Cette définition est une forme normalisée de l'histogramme. On pourrait ne pas diviser les n_k par n, et alors p prendrait ses valeurs dans l'intervalle $[0, n]$.

1 AMELIORATION D'IMAGES

Elle s'appuie sur le contraste correspondant à la différence existant entre les pixels de plus fortes intensités habituellement qualifiés de «clairs », et ceux de plus faibles intensités communément appelés pixels « sombres ». Une image faiblement contrastée présente un aspect terne et sans relief où toute la gamme des tonalités se situe autour des valeurs médianes. Une image fortement contrastée écrase les valeurs tonales intermédiaires en augmentant exagérément celles des plages extrêmes [8][15].

Le but de l'amélioration du contraste est d'obtenir une image plus lisible que celle de départ pour une utilisation spécifique, par exemple la mise en évidence ou le renforcement de certains aspects qu'elle renferme.

Les principales méthodes employées pour l'amélioration des images se fondent sur l'une des approches suivantes :

- l'approche spatiale dont les méthodes opèrent directement sur l'ensemble des pixels composant l'image ;
- l'approche fréquentielle ou spectrale, basée sur la modification de la transformée de Fourier de l'image, qui s'appuie sur le théorème de convolution ou théorème de Plancherel.

Soient $f(x, y)$ une fonction de deux variables continue et intégrable sur un domaine D et $\Im(u, v)$ sa transformée de Fourier. Alors le domaine D est appelé *domaine spatial* et l'ensemble des (u, v) est appelé *domaine des fréquences* ou *domaine spectral*.

1.1 Méthodes du domaine spatial

Soit $f : \mathrm{R}\times\mathrm{R} \to \mathrm{R}$ une image. Il s'agit de trouver une transformation $T : \mathrm{R}\to\mathrm{R}$ qui s'appliquera à f pour donner une image contrastée g telle que :

$g : \mathrm{R}\times\mathrm{R} \to \mathrm{R}$

$g(x, y) = T[f(x, y)]$. (2)

La fonction T s'appliquant sur les valeurs des pixels, nous désignerons par r une valeur de pixel de l'image initiale (l'entrée) et par *s* celle de l'image traitée (la sortie).

1.1.1 Méthode du contraste étiré (contrast stretching)

Elle s'utilise dans le cas d'une image faiblement contrastée résultant soit d'un éclairement insuffisant, soit d'un mauvais réglage de la lentille du capteur. Elle a pour effet d'augmenter le contraste de l'image.

Ici la fonction T doit être croissante, affine sur les trois intervalles ([0, k], [k, j], [j, L-1] avec $0 \le k \le j \le L$ où L est le nombre de valeurs distinctes que peuvent prendre les pixels de l'image) et continue sur [0, L-1]. Dans le cas particulier où l'on a $0 < k = j < L$, et T : [0, L-1] $\to$ {0,1}, avec

T (r) = $\begin{cases} 0 & \text{si r} < m \\ 1 & \text{si r} \ge m \end{cases}$ où $m = L/2$ (appelé point seuil), (3)

on obtient une image binaire (noir/ blanc) en résultat.

1.1.2 Méthode de la compression de la chaîne dynamique

On appelle *chaîne dynamique* le nombre d'intensités différentes que peut prendre chaque pixel de l'image. Lorsque la chaîne dynamique de l'image traitée dépasse la capacité d'affichage du moniteur, seules les plus fortes valeurs de l'image traitée seront généralement visibles (dépassement de capacité). Pour pallier ce problème on peut utiliser la transformation T définie par :

$T(r) = C\,Log(1 + |r|)$ (4)

où C est une constante strictement positive et qui permet de réduire la taille de la chaîne dynamique.

1.1.3 Méthode des niveaux de gris par tranche

Cette méthode est plus indiquée lorsque l'on veut mettre en évidence une partie de l'image. Il existe deux approches :

• Celle où les valeurs des pixels du centre d'intérêt sont remplacées par des intensités élevées, et toutes les autres par de plus basses de sorte à obtenir en résultat une image binaire.
T est alors définie de [0, L -1] vers {0, max} par :

$$T\ (\mathrm{r}) = \begin{cases} \max & si\ \ \mathrm{r} \in \mathrm{A} \\ 0 & si\ \ \mathrm{r} \notin \mathrm{A} \end{cases} \qquad (5)$$

où max désigne la valeur de pixel la plus élevée, et A (A $\subset$ [0, L -1]) l'ensemble des intensités des pixels du centre d'intérêt.

• Celle où les valeurs des pixels du centre d'intérêt sont remplacées par des intensités extrêmement élevées, les autres gardant leurs valeurs de départ ; ce qui réduit la chaîne dynamique de l'image en illuminant les pixels du centre d'intérêt.
T est alors définie de [0, L -1] vers {0, max} par :

$$\mathrm{T(r)} = \begin{cases} \max & si\ \ \mathrm{r} \in \mathrm{A} \\ \mathrm{r} & si\ \ \mathrm{r} \notin \mathrm{A} \end{cases} \qquad (6)$$

où max désigne la valeur de pixel la plus élevée, et A (A $\subset$ [0, L -1]) l'ensemble des intensités des pixels du centre d'intérêt.

1.1.4 Méthode de l'égalisation ou de la linéarisation d'histogramme

On calcule d'abord l'histogramme de l'image initiale puis on spécifie celui que l'on désire avoir après traitement, en général un l'histogramme plat. Ensuite l'on cherche la transformation qui permet de passer de l'un à l'autre. Enfin cette transformation est appliquée à l'image initiale pour obtenir l'image améliorée dont l'histogramme est aplati et étalé.

Comme l'on veut contrôler la densité de probabilité des valeurs de pixels de l'image via une transformation T , celle-ci doit être croissante sur [0, 1] et ses valeurs comprises entre 0 et 1. Pour y parvenir, l'on ramène l'intervalle des valeurs des pixels de l'image de départ à l'intervalle [0, 1] par une division par L -1.

Si p_r est la densité de probabilité de l'image originale et p_s celle de l'image transformée, alors T est liée à p_r par :

$$T\ (\mathrm{r}) = \int_0^{\mathrm{r}} \mathrm{p_r}(x)dx \qquad (7)$$

$$\text{ou}\quad T\ (r_k) = \sum_{\mathrm{j}=0}^{\mathrm{k}} \frac{n_{\mathrm{j}}}{n} = \sum_{\mathrm{j}=0}^{k} \mathrm{p_r}\ (\mathrm{r_j}) \qquad (8)$$

$0 \le r_{\mathrm{j}} \le 1$ et k = 0, 1,…, L -1.

Les égalités (7) et (8) correspondent aux expressions de $T(r)$ dans les cas continu et discret. On montre que p_s (s) = 1 ; p_s correspond donc à une densité de probabilité uniforme. L'histogramme de l'image transformée s'étale alors sur l'intervalle [0, L -1], ce qui traduit un relèvement de contraste.

1.1.5 Méthode du rehaussement local

On définit un voisinage carré $n \times n$ avec n impair, sur lequel on applique la méthode de l'égalisation d'histogramme. En déplaçant le centre du voisinage de pixel en pixel de sorte à couvrir toute l'image, on réalise la méthode du rehaussement local qui a pour conséquence la mise en évidence des détails de l'image.

On peut également appliquer la formule suivante sur chaque pixel *(x, y)* :

$$g(x, y) = A(x, y).[\, f(x, y) - m(x, y)] + m(x, y) \qquad (9)$$

où $A(x, y) = k \dfrac{M}{s\,(x, y)}$, (0<k<1)

$m(x, y)$ la moyenne des pixels du voisinage, $s\,(x, y)$ leur écart type et M la moyenne des valeurs des pixels de l'image entière de départ.

1.2 Méthodes du domaine des fréquences

Dans le cas de ces méthodes on ne s'intéresse plus aux valeurs des pixels, mais à leur transformée de Fourier. Leur procédure d'implémentation s'articule autour des trois points suivants :

- le calcul de la transformée de Fourier $\Im(u, v)$ de l'image $f(x, y)$,
- la multiplication par un filtre de transfert $H(u, v)$,
- La transformation inverse de Fourier du résultat.

Signalons que ces méthodes sont plus difficiles à implémenter par rapport aux précédentes.

1.2.1 Filtrage passe-bas

Il permet d'atténuer les hautes fréquences de $\Im(u, v)$.

- *Le filtre idéal :*

$$H(u, v) = \begin{cases} 1 & \text{si } D(u, v) \le D_0 \\ 0 & \text{si } D(u, v) > D_0 \end{cases} \qquad (10)$$

où D_0 est strictement positif et $D(u, v) = (u^2 + v^2)^{1/2}$.

Toutes les fréquences situées à l'extérieur du cercle de centre O et de rayon D_0 disparaissent et celles situées à l'intérieur de ce cercle sont conservées.

- Dans la pratique, on utilise un filtre dénommé « butterworth »

$$H(u, v) = \frac{1}{1 + [D(u,v)/D_0\,]^{2n}} \qquad (11)$$

avec les mêmes notations que précédemment.

1.2.2 Filtrage passe-haut

Il atténue les composantes basses fréquences sans perturber les informations hautes fréquences de la transformée de Fourier de l'image.

- *Le filtre idéal :*

$$H(u,v) = \begin{cases} 0 & \text{si } D(u, v) \le D_0 \\ 1 & \text{si } D(u, v) > D_0 \end{cases} \qquad (12)$$

où D_0 est strictement positif et $D(u,v) = (u^2 + v^2)^{1/2}$.

Toutes les fréquences situées à l'intérieur du cercle de centre O et de rayon D_0 disparaissent tandis que toutes les autres sont conservées.

- Dans la pratique, on utilise un filtre baptisé « butterworth »

$$H(u,v) = \frac{1}{1 + [D_0 / D(u, v)]^{2n}} \qquad (13)$$

avec les mêmes notations que précédemment.

1.2.3 Filtrage homomorphique

Ici l'on choisit d'écrire la fonction image comme produit de deux fonctions, c'est-à-dire $f(x,y) = i(x,y)r(x,y)$ où $i(x,y)$ est la composante d'éclairement et $r(x,y)$ la composante de réflectance. Le filtrage homomorphique consiste à :

- Prendre le logarithme de f

$$z(x,y) = \ln[f(x,y)] = \ln[i(x,y)] + \ln[r(x,y)]$$

- Appliquer la transformation de Fourier au résultat

$\Im\{z(x,y)\} = \Im\{\ln i(x,y)\} + \Im\{\ln r(x,y)\}$ que l'on écrit $Z(u,v) = I(u,v) + R(u,v)$ avec $Z(u,v) = \Im\{z(x,y)\}$, $I(u,v) = \Im\{\ln i(x,y)\}$, $R(u,v) = \Im\{\ln r(x,y)\}$

- Multiplier le résultat par un filtre de transfert $H(u,v)$

$$S(u,v) = H(u,v)I(u,v) + H(u,v)R(u,v)$$

- Appliquer la transformation inverse de Fourier au résultat obtenu

$$s(x,y) = \Im^{-1}\{S(u,v)\}$$
$$= \Im^{-1}\{H(u,v)I(u,v)\} + \Im^{-1}\{H(u,v)R(u,v)\}$$

que l'on écrit $s(x,y) = i'(x,y) + r'(x,y)$ avec $i'(x,y) = \Im^{-1}\{H(u,v)I(u,v)\}$, $r'(x,y) = \Im^{-1}\{H(u,v)R(u,v)\}$

- Prendre l'exponentielle

$$g(x,y) = \exp[s(x,y)]$$
$$= \exp[i'(x,y)].\exp[r'(x,y)]$$
$$= i_0(x,y)r_0(x,y)$$

où $i_0(x, y = \exp[i'(x,y)]$ et $r_0(x,y) = \exp[r'(x,y)]$.

2 RESTAURATION D'IMAGES

La restauration est nécessaire lorsque l'image est dégradée. Elle a pour but de corriger les distorsions introduites lors des étapes d'acquisition ou de transmission des images. En supposant que l'image ait été dégradée, on va tenter de lui redonner sa qualité originelle en utilisant ce que l'on sait sur la dégradation qu'elle a subie. Cette approche implique donc la définition d'un critère de qualité que l'on cherchera à optimiser.

Le processus de restauration débute avec la modélisation de la dégradation subie par l'image. On procède ensuite à l'application du modèle inverse de la dégradation sur l'image à restaurer, ce qui permet d'obtenir l'image originelle.

2.1 Suppression du bruit

Le bruit est l'ensemble des parasites présents dans l'image. Il peut être dû au système d'acquisition de l'image, aux erreurs de connexion, ou à un mauvais calibrage de la sensibilité des capteurs. Le bruit peut être gaussien (électronique), ou impulsionnel [13].

La suppression du bruit est l'une des premières étapes du traitement d'images. Le traitement d'une image sans bruit est plus simple que celui d'une image bruitée. Pour éliminer le bruit, on procède à un filtrage de l'image.

2.1.1 Filtrage linéaire

Le filtrage linéaire est très utile dans le cas où l'on ne dispose à priori d'aucune connaissance sur l'image à traiter. Il utilise des algorithmes de faible complexité dont les résultats sont satisfaisants sur la plupart des types d'images. Il consiste à remplacer chaque valeur de pixel par une combinaison linéaire des intensités de ses voisins. Les coefficients de cette combinaison linéaire, fournis par la réponse impulsionnelle du filtre, sont représentés par une matrice appelée *masque de convolution* qui donne son nom au filtre.

- *Le filtre « moyenne »*

On remplace chaque valeur de pixel par la moyenne arithmétique de celles de ses voisins. Son masque 3×3 de convolution peut être :

$$\frac{1}{8}\begin{pmatrix}1&1&1\\1&0&1\\1&1&1\end{pmatrix}$$

- *Le filtre « passe-bas » (ou pondération bidimensionnelle de Gauss)*

Il permet d'amortir les parasites et d'interpoler les données manquantes ou endommagées. L'un des masques 3×3 utilisé se présente comme suit :

$$\frac{1}{16}\begin{pmatrix}1&2&1\\2&4&2\\1&2&1\end{pmatrix}$$

On peut également utiliser le masque 3×3 :

$$\frac{1}{9}\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix},$$

ou le masque 5×5 (composé uniquement de 1) multiplié par $\frac{1}{25}$, le masque 7×7 (composé uniquement de 1) multiplié par $\frac{1}{49}$.

- *Le filtre « passe-haut »*

Il élimine les termes de fréquence nulle. Pour l'implémenter, on utilise le masque 3×3 donné ci-après :

$$\frac{1}{9}\begin{pmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{pmatrix}$$

2.1.2 Filtrage non linéaire

- *Le filtrage médian*

Il élimine les pics isolés c'est-à-dire correspondant au bruit de type « salt ». Son principe est simple :

- Sur une fenêtre de $n \times n$ pixels (voisinage d'un pixel) avec n impair, classer les pixels par ordre croissant d'intensité,
- Affecter la valeur de la médiane au pixel central,
- Appliquer cette méthode à toute l'image.

- *La moyenne d'images*

Etant donnée une image bruitée $g(x,y) = f(x,y) + h(x,y)$ où $f(x,y)$ est l'image originelle et $h(x,y)$ le bruit (supposé de moyenne nulle), il s'agit de réduire les effets du bruit en ajoutant à $g(x,y)$ un ensemble d'images bruitées $\{g_i(x,y)\}$.

On peut montrer que $\bar{g}(x,y) = \frac{1}{M}\sum_{i=1}^{M} g_i(x,y)$ (15)

et que $s^2_{\bar{g}}(x,y) = \frac{s^2_h(x,y)}{M}$, (16)

c'est-à-dire que la variabilité des valeurs des pixels diminue quand M, nombre d'images moyennées, augmente.

2.2 Modèle de la dégradation

On assimile le processus de dégradation à un opérateur H, et l'on écrit l'image dégradée sous la forme :

$g(x,y) = H[f(x,y)] + h(x,y)$ (17)

où η représente le bruit.

Les méthodes de restauration d'images [5] sont alors basées sur la recherche de la fonction f, l'opérateur H et le bruit η étant connus.

- *Dans le domaine continu*

Si le bruit est nul ($h(x, y) = 0$) et l'opérateur H linéaire, alors l'égalité (17) s'écrit :

$$g(x, y) = \iint_{R^2} f(a, b)h(x, a, y, b)dadb \qquad (18)$$

avec $h(x,\mu,y,b) = H[d(x-\mu,y-b)]$,

qui est une équation intégrale de Fredholm de première espèce en f [16].

Si le bruit est nul et H un invariant spatial c'est-à-dire $H\ [d(x-\mu,y-b)] = h(x-\mu,y-b)$, alors l'égalité (17) s'écrit :

$$g(x, y) = \iint_{R^2} f(a, b)h(x - a, y-b)dadb \quad , \qquad (19)$$

qui est une équation de convolution en f [9].

- *Dans le domaine discret*

En supposant le bruit nul, l'égalité (17) s'écrit :

$$g_e(x, y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_e(m,n)h_e(x-m,y-n) \qquad (20)$$

x =0,1,…,M-1 ; y =0,1,…,N-1

où f_e et h_e sont les fonctions étendues de f et h supposées périodiques.

La formule (18) peut s'écrire sous forme matricielle de la façon suivante :

$$\mathbf{g = H f} \qquad (21)$$

où **H** est une matrice MN×MN circulaire par blocs, h_e invariant spatial.

Par exemple pour une image 256×256, la matrice H aura une taille de 262144 × 262144 et la recherche de f nécessitera la résolution d'un système de 262144 équations à 262144 inconnues.

Remarque :

Dans l'équation (17) si l'opérateur H est l'identité alors on a le modèle d'une image simplement bruitée et la restauration se résume à la suppression du bruit. Ainsi la suppression du bruit peut être considérée comme une technique de restauration.

2.3 Approche algébrique de résolution

On considère une image dégradée bruitée avec un opérateur de dégradation isotrope :

$$g_e(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_e(m,n)h_e(x-m,y-n) + h_e(x,y)\,, \qquad (22)$$

$x = 0, 1, \ldots, M-1$; $y = 0, 1, \ldots, N-1$
qui s'écrit sous forme matricielle : $\mathbf{g = H f + n}$ (23)
où **f, g** et **n** sont des matrices colonnes MN×1 et **H** une matrice MN×MN circulaire par blocs.

L'approche algébrique consiste à chercher une approximation $\hat{f}$ de f qui minimise un critère de performance prédéfini [1] [3] [4].

2.3.1 Restauration sans contrainte

On réécrit (23) sous la forme $\mathbf{n = g - H f}$ (23 bis)
et l'on pose : $J(\mathbf{f}) = \| \mathbf{g - H f} \|^2$ (24)
La fonction $\hat{f}$ qui minimisera J est la solution de l'équation $\frac{\partial J(f)}{\partial f} = 0$; elle s'exprime sous la forme $\hat{f} = \mathbf{H^{-1}g}$ (25).

2.3.2 Restauration avec contrainte

Dans ce cas $J(\mathbf{f}) = \|\mathbf{Qf}\|^2 + \propto(\| \mathbf{g - H f} \|^2 - \| \mathbf{n} \|^2)$ (26)
où **Q** est un opérateur linéaire sur **f** et $\propto$ un multiplicateur de Lagrange. Dans [15] on montre que la solution recherchée vaut :

$\hat{f} = \mathbf{(H^T H} + \gamma \mathbf{Q^T Q)^{-1} H^T g}$ (27)
avec $\gamma = \propto^{-1}$.

2.3.3 Restauration sous contrainte de régularité

On pose $J(f) = \| g - H f\|^2 + \lambda\phi(f)$ (28)
où $\phi(f)$ est un terme de régularisation qui dépend généralement du gradient de f ou de la matrice des dérivées d'ordre supérieur de f [14]. Le problème de retrouver f à partir de g est alors formulé comme celui de retrouver f qui minimise l'énergie $J(f)$.

Si $\phi(f) = |\nabla f|^2$ est quadratique, on a la régularisation de Tikhonov qui, malheureusement, pénalise les discontinuités.

2.4 Filtrage inverse

On suppose que l'image est carrée et **H** diagonalisable ; alors l'égalité (25) devient

$\mathbf{W^{-1}}\ \hat{f} = \mathbf{D^{-1}\, W^{-1}\, g}$ (29)
où **D** est une matrice diagonale et **W** une matrice de passage .

On montre que l'égalité (29) vaut :

$$\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)} \quad ; \; u, v = 0, 1, 2, \ldots, N-1 \qquad (30)$$

G(u, v) est donc multipliée par l'inverse du filtre H(u, v), d'où le nom de la méthode. En prenant la transformation inverse de Fourier de $\hat{F}(u, v)$, on retrouve $\hat{f}$.

2.5 Filtrage de Wiener

On considère le filtre :

$$\frac{1}{H(u, v)} \frac{|H(u, v)|^2}{|H(u, v)|^2 + g[S_n(u, v)/S_f(u, v)]}$$ où $S_f(u,v)$ et $S_n(u,v)$ sont les densités spectrales respectives de $f_e(x, y)$ et $\eta_e(x, y)$.

Si $\gamma = 1$ alors le filtre ci-dessus est connu sous le nom de filtre de Wiener .

Si $S_n(u,v) = 0$ (absence de bruit) alors on retrouve le filtre inverse.

2.6 Transformations géométriques

A la différence des techniques évoquées jusqu'ici, les transformations géométriques modifient en général les relations spatiales entre les pixels d'une image. Elles sont le plus souvent appelées transformations en *feuille de caoutchouc*, car comparables à ce que l'on obtiendrait d'une image imprimée sur une feuille de caoutchouc qu'on étirerait suivant certaines règles bien définies.

En termes de traitement d'images numériques, une transformation géométrique consiste à réaliser deux opérations de base :

- Une transformation spatiale qui définit le réarrangement des pixels sur l'image plane ;
- Une interpolation des valeurs des pixels qui ré-attribue différentes valeurs à chacun des pixels de l'image transformée spatialement.

2.7 Techniques utilisant les équations aux dérivées partielles (EDP)

Les méthodes précédentes fournissent des résultats qui présentent des oscillations près des discontinuités. Pour contourner cette difficulté, les nouvelles approches posent le problème de la restauration d'images comme un problème de régularisation avec l'introduction des EDP [14]. L'approche classique utilise une EDP linéaire où l'image restaurée *u* peut s'écrire comme produit de convolution de l'image bruitée *v* avec un opérateur de lissage G :

$$u = G * v \qquad (31)$$

Si l'on suppose G gaussien, l'équation (31) peut alors se mettre sous la forme d'une équation aux dérivées partielles de type parabolique linéaire :

$$\begin{cases} \dfrac{\partial u}{\partial t}(x, y, t) = u_{xx}(x, y, t) + u_{yy}(x, y, t) \\ u(x, y, 0) = u_0(x, y) \end{cases} \qquad (32)$$

Cette EDP permet une diffusion isotrope, ce qui présente des inconvénients notamment au niveau de la qualité visuelle de l'image. Pour contourner cette difficulté, on utilise le modèle de Pérona et Malik [11][12] donné par :

$$\begin{cases} \dfrac{\partial u}{\partial t}(x,y,t) = div(c(\left|\nabla u(x,y,t)\right|)\nabla u(x,y,t)) \\ u(x,y,0) = u_0(x,y) \end{cases} \qquad (33)$$

où div et ∇ représentent respectivement les opérateurs de divergence et de gradient par rapport aux variables spatiales et où la fonction *c(.)* est décroissante.

L'équation (33) est une EDP non linéaire. Mais ce modèle s'avère inefficace dans les zones où le bruit présente de grosses discontinuités. Pour surmonter ces difficultés d'autres modèles sont présentés dans [10].

DEUXIEME PARTIE

Pseudo code, processus d'implémentation et code source de quelques algorithmes de pré-traitement d'images

3 Pseudo codes des quelques algorithmes

3.1 Filtre médian

Procédure f_median ;
/* Applique le filtre médian sur des données images prises en entrée */
Données en entrée :
source : pointeur ; /* pointe sur le tableau correspondant à l'image source */
ligne : entier positif ; /* nombre de lignes de l'image en entrée */
colonne : entier positif ; /* nombre colonnes de l'image en entrée */
Résultats :
destination : pointeur ; /* pointeur sur le tableau correspondant à l'image traitée */
Variables de travail :
pt : pointeur ; /* pointeur sur un tableau de valeurs correspondant à celles d'un voisinage d'un pixel en entrée*/
Enchaînement des opérations :
Début
Allouer(pt) ;
Traiter_les_elts_non_en_bordure (source, destination, ligne, colonne) ;
Traiter_les_elts_en_bordure (source, destination, ligne, colonne) ;
Libérer(pt) ;
Retourner(destination) ;
Fin.

Raffinement de Traiter_les_elts_non_en_bordure ;
/* Applique le filtre médian sur les éléments les plus internes de l'image en entrée */
Données en entrée :
Source : pointeur ; /* pointeur sur l'image en entrée */
ligne : entier positif ; /* nombre de lignes de l'image en entrée */
colonne : entier positif ; /* nombre colonnes de l'image en entrée */
Données en sortie :
Destination : pointeur ; /* pointe sur l'image en sortie */
Variables de travail :
i, j : entier positif ;
Enchaînement des opérations :
Début
i←2 ; /* permet de parcourir les lignes de l'image */
j←2 ; /* permet de parcourir les colonnes de l'image */
Tant que *i < derniere_ligne* faire
Tant que *j < dernière_colonne* faire
Ranger dans le tableau pointé par *pt* les éléments

du voisinage 3×3 du pixel *(i,j)* de l'image pointée par *source* ;

Rechercher et ranger la valeur médiane en position *(i,j)*
dans le tableau pointé par *destination* ;
j←j+1 ;

Fin tant que
i←i+1

Fin tant que

Fin

Raffinement de Traiter_les_elts_en_bordure ;
/* Applique le filtre médian sur les éléments en bordure de l'image prise en entrée */
Données en entrée :
Source : pointeur ; /* pointeur sur l'image en entrée */
ligne : entier positif ; /* nombre de lignes de l'image en entrée */
colonne : entier positif ; /* nombre colonnes de l'image en entrée */
Données en sortie :
Destination : pointeur ; /* pointe sur l'image en sortie */
Variables de travail :
i, j : entier positif ;
Enchaînement des opérations :
début
/* Pour les éléments extrêmes : première et dernière lignes */
Ranger dans le tableau pointé par *pt* les éléments
du voisinage 2×3 du pixel *(i,j)* de l'image pointée par *source* ;

Rechercher et ranger la valeur médiane en position *(i,j)*
dans le tableau pointé par *destination* ;

/* Pour les éléments extrêmes : première et dernière colonne */
Ranger dans le tableau pointé par *pt* les éléments
du voisinage 3×2 du pixel *(i,j)* de l'image pointée par *source* ;

Rechercher et ranger la valeur médiane en position *(i,j)*
dans le tableau pointé par *destination* ;

/* Pour les quatre coins */
Ranger dans le tableau pointé par *pt* les éléments
du voisinage 2×2 du pixel *(i,j)* de l'image pointée par *source* ;

Rechercher et ranger la valeur médiane en position *(i,j)*
dans le tableau pointé par *destination* ;
Fin

Fonction médiane ;
/* Recherche la valeur médiane d'un tableau d'entiers */
Données en entrée :
source : pointeur ; /* pointe sur le tableau d'entiers */

taille : entier positif ; /* taille du tableau*/
Résultats :
v_median : entier ; /* valeur médiane du tableau*/
Enchaînement des opérations :
Début
Ranger les éléments du tableau dans l'ordre croissant ;
Si taille est impaire alors
v_median=élément du milieu du tableau
sinon
v_median=moyenne des deux éléments centraux
Finsi ;
Retourner (v_median)
Fin

3.2 Filtre « moyenne »

Procédure f_moyen ;
/* Applique le filtre « moyenne » sur l'image en entrée */
Données en entrée :
source : pointeur ; /* pointe sur le tableau correspondant à l'image source */
ligne : entier positif ; /* nombre de lignes de l'image en entrée */
colonne : entier positif ; /* nombre colonnes de l'image en entrée */
Résultats :
destination : pointeur ; /* pointeur sur le tableau correspondant à l'image traitée */
Enchaînement des opérations :
Début
Allouer(pt) ;
Traiter_les_elts_non_en_bordure (source, destination, ligne, colonne) ;
Traiter_les_elts_en_bordure (source, destination, ligne, colonne) ;
Libérer(pt) ;
Retourner(destination) ;
Fin.

Remarque :
Les procédures Traiter_les _elts_non_en_bordure et Traiter_les_elts_en_bordure
ont la même structure que précédemment. Il suffit de remplacer médiane par moyenne.

Fonction demi_moyenne ;
/* Recherche la moyenne des éléments d'un tableau d'entiers en utilisant comme diviseur taille-1 */
Données en entrée :
source : pointeur ; /* pointe sur le tableau d'entiers */
taille : entier positif ; /* taille du tableau*/
Résultats :
v_moyen : entier ; /* valeur moyenne entière du tableau*/
Enchaînement des opérations :
Début

v_moyen= (somme des éléments du tableau pointé par source)/(taille-1) ;
Retourner(v_moyen) ;

Fin

3.3 Filtre « passe_bas »

Procédure f_p_bas ;
/* Applique le filtre « passe_bas » sur l'image en entrée
Elle est semblable à la procédure filtre moyen ; la seule différence est qu'on utilise une fonction moyenne qui calcule la moyenne arithmétique */

Fonction moyenne ;
/* Recherche la moyenne arithmétique des éléments d'un tableau d'entiers */
Données en entrée :
source : pointeur ; /* pointe sur le tableau d'entiers */
taille : entier positif ; /* taille du tableau*/
Résultats :
v_moyen : entier ; /* valeur moyenne entière du tableau*/
Enchaînement des opérations :
Début
v_moyen= (somme des éléments du tableau pointé par source)/taille ;
Retourner(v_moyen) ;
Fin

4 CODES SOURCES ET TESTS D'IMPLEMENTATION

4.1 Le filtre médian

```
   /*=====================================================================*
    *      CE MODULE CONTIENT LES FONCTIONS mediane ET f_median           *
    *=====================================================================*/
/***********************************************************************
  *                                                                      *
  * EN ENTREE: Pointeur sur un tableau d'entiers positifs,               *
  *           taille du tableau.                                         *
  * EN SORTIE: entier positif correspondant … la valeur m,diane des      *
  *           éléments du tableau                                        *
  *                                                                      *
  ***********************************************************************/

 unsigned int mediane(unsigned int *p,unsigned int taille)
 {
   unsigned int i,k; /*Compteurs*/
   unsigned int aux; /*Variable auxiliaire*/
   unsigned int v_median; /*Variable médiane*/
   k=0; /*Mettre le compteur à zéro*/

   /*Ordonner les éléments du tableau*/
   do { for(i=k+1;i<taille;i++)
       if(p[k]>p[i])   /*Si le k ième élément est supérieur au  ième*/
```

```
        { aux=p[k];   /*alors permuter ces deux éléments*/
          p[k]=p[i];
          p[i]=aux;
         }    /*Le k ième élément du tableau est à sa place définitive*/
      k++;     /*On passe à l'élément suivant*/
       }
   while(k<taille-1); /*On s'arrête à l'avant dernier élément*/

   if(taille%2)  v_median = p[(taille-1)/2];
   else v_median = (p[(taille-1)/2]+p[taille/2])/2;

   return(v_median); /*Retourne la valeur médiane*/

 }   /*Fin de la fonction mediane*/



/*************************************************************************
 *                                                                       *
 * EN ENTREE: Pointeur sur des données images,                           *
 *            nombre de lignes et de colonnes de l'image                 *
 * EN SORTIE:  Pointeur sur des données images traitées.                 *
 *                                                                       *
 *************************************************************************/

 unsigned int *f_median(unsigned int *p,unsigned int ligne,
                                    unsigned int colonne)
  {
   unsigned int i,j;  /*Comptent du le nombre de lignes et de colonnes
                                                 de l'image*/
   unsigned int *pt;  /*Pointeur sur un tableau 3*3 de valeurs des pixels
                 voisins du pixel (i,j)*/
   unsigned int *aux; /*Pointeur retourné par la fonction*/
   int m,n;             /*Comptent le nombre de lignes et de colonnes du
                                  voisinage de chaque pixel*/

   /*Alloue de l'espace pour ranger les données filtrées*/
   if((aux=(unsigned int *)calloc(ligne*colonne,sizeof(unsigned
int)))==NULL)
     { printf("Mémoire insuffisante pour ranger les valeurs retournées par"
                                             " f_median\n");
       getch();
       exit(1);
      }

   /*Alloue de l'espace pour ranger les éléments du voisinage de chaque
pixel*/
   pt=(unsigned int *)calloc(9,sizeof(unsigned int));

   /*Eléments les plus internes*/
   for(i=1;i<ligne-1;i++)
    for(j=1;j<colonne-1;j++)
     {
       for(m=-1;m<2;m++)
      for(n=-1;n<2;n++)
       { pt[3*(m+1)+n+1]=p[(i+m)*colonne+j+n];  /*Fait pointer pt sur les
                         neuf éléments du voisinage du pixel (i,j)*/
        }
       aux[i*colonne+j]=mediane(pt,9);
      }
```

```
 /*Eléments de la première ligne sauf les deux extrémités*/
 for(j=1;j<colonne-1;j++)
  {
    for(m=-1;m<1;m++)
     for(n=-1;n<2;n++)
     { pt[3*(m+1)+n+1]=p[(1+m)*colonne+j+n];  /*Fait pointer pt sur les
                       six éléments du voisinage du pixel (0,j)*/
     }
     aux[j]=mediane(pt,6);
   }

/*Eléments de la dernière ligne sauf les deux extrémités*/
 for(j=1;j<colonne-1;j++)
  {
     for(m=-1;m<1;m++)
    for(n=-1;n<2;n++)
     { pt[3*(m+1)+n+1]=p[(ligne-1+m)*colonne+j+n];  /*Fait pointer pt sur
           les six éléments du voisinage du pixel (ligne-1,j)*/
     }
     aux[(ligne-1)*colonne+j]=mediane(pt,6);
   }

/*Eléments de la première colonne sauf les deux extrémités*/
 for(i=1;i<ligne-1;i++)
  {
     for(m=-1;m<2;m++)
    for(n=-1;n<1;n++)
     { pt[2*(m+1)+n+1]=p[(i+m)*colonne+1+n];   /*Fait pointer pt sur les
                six éléments du voisinage du pixel (i,0)*/
     }
     aux[i*colonne]=mediane(pt,6);
  }

/*Eléments de la dernière colonne sauf les extrémités*/
 for(i=1;i<ligne-1;i++)
  {
     for(m=-1;m<2;m++)
    for(n=-1;n<1;n++)
     { pt[2*(m+1)+n+1]=p[(i+1+m)*colonne-1+n]; /*Fait pointer pt sur les
                six éléments du voisinage du pixel (i,colonne-1)*/
     }
     aux[(i+1)*colonne-1]=mediane(pt,6);
   }

/*Coin supérieur gauche*/
  pt[0]=p[0];
  pt[1]=p[1];
  pt[2]=p[colonne];
  pt[3]=p[colonne+1];
  aux[0]=mediane(pt,4);

/*Coin supérieur droit*/
  pt[0]=p[colonne-2];
  pt[1]=p[colonne-1];
  pt[2]=p[2*colonne-2];
  pt[3]=p[2*colonne-1];
  aux[colonne-1]=mediane(pt,4);

/*Coin inférieur gauche*/
  pt[0]=p[(ligne-2)*colonne];
  pt[1]=p[(ligne-2)*colonne+1];
```

```
    pt[2]=p[(ligne-1)*colonne];
    pt[3]=p[(ligne-1)*colonne+1];
    aux[(ligne-1)*colonne]=mediane(pt,4);

  /*Coin inférieur droit*/
    pt[0]=p[(ligne-1)*colonne-2];
    pt[1]=p[(ligne-1)*colonne-1];
    pt[2]=p[ligne*colonne-2];
    pt[3]=p[ligne*colonne-1];
    aux[ligne*colonne-1]=mediane(pt,4);

    free(pt);

  return(aux); /*retourner le pointeur sur les données filtrées*/

 } /*Fin de la fonction f_median*/
```

4.2 Le filtre « moyenne »

```
   /*=====================================================================*
    *      CE MODULE CONTIENT LES FONCTIONS demi_moyenne ET  f_moyen    *
    *=====================================================================*/

/***********************************************************************
 *                                                                     *
 * EN ENTREE: Pointeur sur un tableau à N éléments,
 *            taille du tableau
 * EN SORTIE: Moyenne des N-1 éléments du tableau
 *
 ***********************************************************************/

 unsigned int demi_moyenne(unsigned int *p,unsigned int taille)
 {
   unsigned int i;   /*Compteurs*/
   unsigned int v_moyen; /*Valeur moyenne*/

   v_moyen=0;

   for(i=0;i<taille;i++) /*Parcourir le tableau*/
    { v_moyen+=p[i]; /*Fait la somme des éléments du tableau*/
    }
   v_moyen/=(taille-1);

   return(v_moyen); /*Retourne la "demi_moyennne"*/

 }   /*Fin de la fonction demi_moyenne*/



/***********************************************************************
 *                                                                     *
 * EN ENTREE: Pointeur sur les données images,                         *
 *            nombre de lignes et de colonnes de l'image               *
 * EN SORTIE: Pointeur sur des données images traitées.                *
 *                                                                     *
 ***********************************************************************/

 unsigned int *f_moyen(unsigned int *p,unsigned int ligne,unsigned int
colonne)
```

```
  {
   unsigned int i,j; /*Compteurs*/
   unsigned int *pt; /*Pointe sur un tableau de valeurs des pixels voisins
                  du pixel (i,j)*/
   unsigned int *aux; /*Pointeur retourné par la fonction*/
   int m,n; /*Compteurs*/

   /*Alloue de l'espace pour ranger les données filtrées*/
   if((aux=(unsigned int *)calloc(ligne*colonne,sizeof(unsigned
int)))==NULL)
    { printf("M,moire insuffisante pour ranger les valeurs retournées par"
                                                 " f_median\n");
      getch();
      exit(1);
    }

  /*Alloue de l'espace pour ranger les valeurs des pixels voisins du
                                           pixel (i,j)*/
  pt=(unsigned int *)calloc(9,sizeof(unsigned int));

  /*Eléments les plus internes*/
   for(i=1;i<ligne-1;i++)
    for(j=1;j<colonne-1;j++)
     {
       for(m=-1;m<2;m++)
      for(n=-1;n<2;n++)
       { pt[3*(m+1)+n+1]=p[(i+m)*colonne+j+n];
       }
       pt[4]=0; /*L'élément central est nul*/
       aux[i*colonne+j]=demi_moyenne(pt,9);
     }

  /*Eléments de la première ligne sauf les deux extrémités*/
   for(j=1;j<colonne-1;j++)
    {
       for(m=-1;m<1;m++)
      for(n=-1;n<2;n++)
       { pt[3*(m+1)+n+1]=p[(1+m)*colonne+j+n];
       }
       pt[1]=0; /*Le deuxième élément du voisinage est nul*/
       aux[j]=demi_moyenne(pt,6);
     }

  /*Eléments de la dernière ligne sauf les extrémités*/
   for(j=1;j<colonne-1;j++)
    {
       for(m=-1;m<1;m++)
      for(n=-1;n<2;n++)
       { pt[3*(m+1)+n+1]=p[(ligne-1+m)*colonne+j+n];
       }
       pt[4]=0;  /*Le cinquième élément est nul*/
       aux[(ligne-1)*colonne+j]=demi_moyenne(pt,6);
     }

  /*Eléments de la première colonne sauf les extrémités*/
   for(i=1;i<ligne-1;i++)
    {
       for(m=-1;m<2;m++)
      for(n=-1;n<1;n++)
       { pt[2*(m+1)+n+1]=p[(i+m)*colonne+1+n];
       }
```

```
       pt[2]=0;  /*Le troisième élément est nul*/
       aux[i*colonne]=demi_moyenne(pt,6);
    }

  /*Eléments de la dernière colonne sauf les extrémités*/
   for(i=1;i<ligne-1;i++)
    {
       for(m=-1;m<2;m++)
      for(n=-1;n<1;n++)
       { pt[2*(m+1)+n+1]=p[(i+1+m)*colonne-1+n];
       }
       pt[3]=0;  /*Le quatrième élément est nul*/
       aux[(i+1)*colonne-1]=demi_moyenne(pt,6);
     }

  /*Coin supérieur gauche*/
    pt[0]=0;
    pt[1]=p[1];
    pt[2]=p[colonne];
    pt[3]=p[colonne+1];
    aux[0]=demi_moyenne(pt,4);

  /*Coin supérieur droit*/
    pt[0]=p[colonne-2];
    pt[1]=0;
    pt[2]=p[2*colonne-2];
    pt[3]=p[2*colonne-1];
    aux[colonne-1]=demi_moyenne(pt,4);

  /*Coin inférieur gauche*/
    pt[0]=p[(ligne-2)*colonne];
    pt[1]=p[(ligne-2)*colonne+1];
    pt[2]=0;
    pt[3]=p[(ligne-1)*colonne+1];
    aux[(ligne-1)*colonne]=demi_moyenne(pt,4);

  /*Coin inférieur droit*/
    pt[0]=p[(ligne-1)*colonne-2];
    pt[1]=p[(ligne-1)*colonne-1];
    pt[2]=p[ligne*colonne-2];
    pt[3]=0;
    aux[ligne*colonne-1]=demi_moyenne(pt,4);

    free(pt); /*Libération de l'espace allouer pour ranger les éléments du
                                          voisinage*/

  return(aux);

 } /*Fin de la fonction f_median*/
```

4.3 Le filtre « passe_bas »

```
   /*=================================================================*
    *      CE MODULE CONTIENT LES FONCTIONS moyenne ET  f_p_bas        *
    *=================================================================*/

/*******************************************************************
 *                                                                 *
 * EN ENTREE: Pointeur sur un tableau de données,                  *
 *            taille du tableau                                    *
```

```
 * EN SORTIE: Moyenne des éléments de ce tableau.                        *
 *                                                                       *
 *************************************************************************/

 unsigned int moyenne(unsigned int *p,unsigned int taille)
 {
   unsigned int i;   /*Compteurs*/
   unsigned int v_moyen; /*Valeur moyenne*/

   v_moyen=0;

   for(i=0;i<taille;i++) /*Parcourir le tableau*/
    { v_moyen+=p[i]; /*Faire la somme de ses éléments*/
    }
   v_moyen/=taille;
   return(v_moyen); /*Retourne la valeur moyenne*/

 }   /*Fin de la fonction moyenne*/


/*************************************************************************
 *                                                                       *
 * EN ENTREE: Pointeur sur les données images,                           *
 *            nombre de lignes et de colonnes de l'image                 *
 * EN SORTIE: Pointeur sur des données images traitées.                  *
 *                                                                       *
 *************************************************************************/

 unsigned int *f_p_bas(unsigned int *p,unsigned int ligne,unsigned int
colonne)
 {
   unsigned int i,j; /*compteurs*/
   unsigned int *pt; /*pointe sur les valeurs de pixels d'un voisinage du
                                                    pixel (i,j)*/
   unsigned int *aux; /*Valeur retournée par la fonction*/
   int m,n; /*Compteur*/

   /*Alloue de l'espace pour ranger les données filtrées*/
   if((aux=(unsigned int *)calloc(ligne*colonne,
                                        sizeof(unsigned int)))==NULL)
    { printf("Mémoire insuffisante pour ranger les valeurs retournées par"
                                             " f_median\n");
      getch();
      exit(1);
    }

   /*Allocation d'espace pour ranger les valeurs des pixels voisins
                                           du pixel (i,j)*/
   pt=(unsigned int *)calloc(9,sizeof(unsigned int));

  /*Eléments les plus internes*/
   for(i=1;i<ligne-1;i++)
    for(j=1;j<colonne-1;j++)
     {
       for(m=-1;m<2;m++)
      for(n=-1;n<2;n++)
       { pt[3*(m+1)+n+1]=p[(i+m)*colonne+j+n];
       }
       aux[i*colonne+j]=moyenne(pt,9);
     }
```

```
/*Eléments de la première ligne sauf les extrémités*/


 for(j=1;j<colonne-1;j++)
  {
     for(m=-1;m<1;m++)
    for(n=-1;n<2;n++)
     { pt[3*(m+1)+n+1]=p[(1+m)*colonne+j+n];
     }
     aux[j]=moyenne(pt,6);
   }

/*Eléments de la dernière ligne sauf les extrémités*/
 for(j=1;j<colonne-1;j++)
  {
     for(m=-1;m<1;m++)
    for(n=-1;n<2;n++)
     { pt[3*(m+1)+n+1]=p[(ligne-1+m)*colonne+j+n];
     }
     aux[(ligne-1)*colonne+j]=moyenne(pt,6);
   }

/*Eléments de la première colonne sauf les extrémités*/
 for(i=1;i<ligne-1;i++)
  {
     for(m=-1;m<2;m++)
    for(n=-1;n<1;n++)
     { pt[2*(m+1)+n+1]=p[(i+m)*colonne+1+n];
     }
     aux[i*colonne]=moyenne(pt,6);
  }

/*Eléments de la dernière colonne sauf les extrémités*/
 for(i=1;i<ligne-1;i++)
  {
     for(m=-1;m<2;m++)
    for(n=-1;n<1;n++)
     { pt[2*(m+1)+n+1]=p[(i+1+m)*colonne-1+n];
     }
     aux[(i+1)*colonne-1]=moyenne(pt,6);
   }

/*Coin supérieur gauche*/
  pt[0]=p[0];
  pt[1]=p[1];
  pt[2]=p[colonne];
  pt[3]=p[colonne+1];
  aux[0]=moyenne(pt,4);

/*Coin supérieur droit*/
  pt[0]=p[colonne-2];
  pt[1]=p[colonne-1];
  pt[2]=p[2*colonne-2];
  pt[3]=p[2*colonne-1];
  aux[colonne-1]=moyenne(pt,4);

/*Coin inférieur gauche*/
  pt[0]=p[(ligne-2)*colonne];
  pt[1]=p[(ligne-2)*colonne+1];
  pt[2]=p[(ligne-1)*colonne];
  pt[3]=p[(ligne-1)*colonne+1];
```

```
    aux[(ligne-1)*colonne]=moyenne(pt,4);

  /*Coin inférieur droit*/
    pt[0]=p[(ligne-1)*colonne-2];
    pt[1]=p[(ligne-1)*colonne-1];
    pt[2]=p[ligne*colonne-2];
    pt[3]=p[ligne*colonne-1];
    aux[ligne*colonne-1]=moyenne(pt,4);

    free(pt); /*Libération de l'espace allouer*/

  return(aux);

 } /*Fin de la fonction f_median*/
```

4.4 Le zoom d'une image

```
/*=========================================================================*
 *      PROGRAMME DE VISUALISATION DES IMAGES                              *
 *=========================================================================*/

#include <graphics.h>

/*************************************************************************
 *                                                                       *
 * EN ENTREE: Pointeur sur des données images,                           *
 *            nombre de lignes et de colonnes de l'image
 * EN SORTIE: Image de dimensions triplées                               *
 *                                                                       *

*************************************************************************/

void zoom_3(unsigned int *p,unsigned int ligne,unsigned int colonne,int
x0,int y0)
 {
  unsigned int i,j; /*Compte les lignes et colonnes de l'image*/
  unsigned int k;   /*Compte les données images*/
  int m,n;          /*Permettent de quadrupler chaque pixel*/

  j=0;i=0;k=0;      /*Initialisation*/
  do
   { for(n=0;n<3;n++)                    /*Affichage d'un*/
      for(m=0;m<3;m++)                   /*carré 3*3 de pixels*/
       { putpixel(x0+i+m,y0+j+n,*(p+k));
        if( (x0+i+m)>getmaxy() || (y0+j+n)>getmaxx())
         { printf("Débordement d'écran graphique pour le triple "
                                     "de l'image\n");
          getch();
          exit(1);
         }
       }
     i+=3; /*Saute une colonne*/
     k++;  /*Passe à la donnée image suivante*/

     if(!(k%colonne))   /*Si l'on a lu une ligne de données images alors*/
      { j+=3;       /*saute deux lignes de l'image que l'on est en train*/
      i=0;            /*d'afficher et revient à la première colonne*/
      }
   } while(k<ligne*colonne); /*Sort si l'on a lu la dernière donnée image*/
```

```
 } /*Fin de la fonction zoom_3*/
```

4.5 Génération des données

```
 /*************************************************************************
  *                                                                       *
  *  Crée un tableau de données de type caractère non signé de dimensions *
  *  "ligne" lignes sur "colonne" colonnes                                *
  *                                                                       *
 *************************************************************************/

  unsigned int *creation(unsigned int *p,unsigned int ligne,unsigned int
colonne)
   {
    unsigned int i;
    randomize();
    for (i=0;i<ligne*colonne;i++)
     { *(p+i)=random(16);
      if (*(p+i)<15)  *(p+i)=3;
      }
     return(p);
   }
```

4.6 Affichage des données

```
  /*********************************************************************
   * Affiche les données générées sur l'écran en mode texte            *
   *                                                                   *
   *********************************************************************/

 void afficher(unsigned int *p,unsigned int ligne,unsigned int colonne)
  {
    unsigned int i;
    i=0;
    printf("\n");
    do { printf("%3d",*(p+i));
       i++;
       if(!(i%colonne))   printf("\n");
      }
    while(i<ligne*colonne);
    printf("\n");
   }
```

4.7 Passage en mode graphique

```
/*======Fonction de passage en mode graphique: VGA 16 couleurs===*/
void initialiser(void)
 {
  int graphdriver,graphmode,erreur;
  graphdriver=DETECT;
  initgraph(&graphdriver,&graphmode,"c:\\tc\\bgi");
  erreur=graphresult();
  if(erreur<0)
    { printf("Erreur dans initgraph :%s.\n",grapherrormsg(erreur));
      exit(1);
    }
 }
```

4.8 Fonction principale

```
/*TEST DES MODULES */

/*****************************************Fichiers à inclure*/
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <graphics.h>
#include <stdlib.h>
#include <time.h>

/*****************************************Modules à inclure*/
#include "c:\imsimes\calcul.c"
#include "c:\imsimes\graphe.c"


/*****************************************Variables globales*/
unsigned int *p1,*p2,*p3,*p4; /*Pointeurs sur des données images*/
unsigned int nb_lig,nb_col;   /*Dimensions de l'image*/
unsigned int i;               /*Compteur*/

main()
{
 clrscr();
 printf("\"N'affiche pas des images de plus de 80*80\"\n");
 printf("Donnez les dimensions (en pixels) de l'image:\n");
 do { printf("Hauteur de l'image en pixels: ");
      scanf("%d",&nb_lig);
    } while(nb_lig<2 || nb_lig >75);

 do { printf("Largeur de l'image en pixels: ");
      scanf("%d",&nb_col);
    } while(nb_col<2 || nb_col >75);

/*****************************************Allocation de mémoire*/
 if((p1=(unsigned int *)calloc(nb_lig*nb_col,sizeof(unsigned int)))==NULL)
  { printf("Pas de m,moire pour cr,er les donn,es\n");
    getch();
    exit(1);
  }

 p1=creation(p1,nb_lig,nb_col); /*Données initiales*/
 p2=f_median(p1,nb_lig,nb_col); /*Données retournées par le filtre médian*/
 p3=f_moyen(p1,nb_lig,nb_col); /*Données retournées par le filtre
"moyenne"*/
 p4=f_p_bas(p1,nb_lig,nb_col); /*Données retournées par le filtre
"passe_bas"*/

 clrscr();

 /*********************************Affichage des données à l'écran*/
 printf("\n Données initiales \n");
 afficher(p1,nb_lig,nb_col);
 getch();

 printf("Résultats du filtre median\n");
 afficher(p2,nb_lig,nb_col);
```

```
 getch();

 printf("Résultats du filtre moyen\n");
 afficher(p3,nb_lig,nb_col);
 getch();

 printf("Résultats du filtre passe_bas\n");
 afficher(p4,nb_lig,nb_col);
 getch();

 initialiser(); /*Passage en mode graphique*/
 cleardevice();

 /**********************************Affichage des histogrammes*/
 affhist(p1,nb_lig*nb_col,"( Données initiales )");
 getch();
 cleardevice();

 affhist(p2,nb_lig*nb_col,"( Filtre médian )");
 getch();
 cleardevice();

 affhist(p3,nb_lig*nb_col,"( Filtre moyenne )");
 getch();
 cleardevice();


 affhist(p4,nb_lig*nb_col,"( Filtre passe_bas )");
 getch();
 cleardevice();

 zoom_3(p1,nb_lig,nb_col,0,0);
 zoom_3(p2,nb_lig,nb_col,3*nb_col+5,0);
 zoom_3(p3,nb_lig,nb_col,0,3*nb_lig+5);
 zoom_3(p4,nb_lig,nb_col,3*nb_col+5,3*nb_lig+5);
 getch();

 closegraph();
 free(p1);
 free(p2);
 free(p3);
 free(p4);
}
```

5 GUIDE D'UTILISATION

Le code source ci-dessus est composé de deux modules. Pour compiler le programme principal, nommé TEST_Z.c, il faut compiler et relier ces modules. Vous devez donc :

1. Créer un fichier calcul.c qui contiendra les fonctions et procédures creation, afficher, mediane, f_median, moyenne, f_p_bas, demi_moyenne et f_moyen. ;
2. Créer un fichier graphe pour la fonction initialiser, zoom_3 ;
3. Compiler TEST_Z.c ;
4. Faire l'édition de liens et exécuter.

On peut modifier la boucle for de la fonction creation dans le but d'avoir des données différentes.

TROISIEME PARTIE
Résultats des tests sur des tableaux de pixels

Test1 :

nb_lig=20, nb_col=20
Dans creation, remplacer la boucle « for » par :

```
for (i=0;i<ligne*colonne;i++)
 {  *(p+i)=random(16) ;
        if(*(p+i)<15)  *(p+i)=3;
 }
```

On génère alors un tableau 20 ×20 des valeurs composé de 3 et 15.
Voici les résultats :

Données intiales

3	3	3	3	3	3	3	3	3	3	3	3	15	15	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	15	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	15	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	15	3	3	3	3	3	3	3	15	3	3	3	15	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	15
3	3	3	3	3	15	3	3	3	3	3	3	3	3	3	15	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	15	3	3	3	3	3	3	3	15	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	15	3	3
3	3	3	3	3	3	3	3	3	3	3	15	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	15	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	15	3	15	3	15	15	3	3	15	3	15	3	3	3	3	3	15	3
3	3	15	3	3	15	3	3	3	15	15	3	3	3	3	3	3	3	3	3

Les « 15 » sont peu nombreux et dispersés ; ils peuvent être assimilés à du bruit. On suppose alors qu'on a une image de couleur unique sur laquelle figurent des tâches. L'idée est alors d'effacer ces tâches.

ésultats du filtre median

15	9	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
9	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Après passage du filtre médian, seul le coin supérieur gauche est entaché.

Résultats du filtre moyen

7	3	7	5	5	3	3	3	3	3	3	3	3	3	3	3	3	5	5	7
5	4	6	3	4	3	3	4	4	4	3	3	3	3	3	3	3	4	3	5
3	3	4	4	4	3	3	4	3	4	3	3	3	3	3	3	3	4	4	5
3	3	3	3	3	3	3	4	4	4	3	3	3	4	4	4	3	4	4	5
3	3	3	3	4	4	4	3	3	4	4	4	3	4	3	4	3	4	3	5
3	3	3	3	4	3	4	3	3	4	3	4	3	4	4	6	4	6	4	5
3	4	4	4	4	4	4	3	3	4	4	4	3	3	3	4	3	4	3	3
5	6	3	4	3	3	3	3	3	4	4	4	3	3	3	4	4	6	4	5
3	6	4	4	3	3	3	4	4	6	3	4	3	3	3	3	4	6	4	5
5	4	3	4	4	4	3	4	3	6	4	4	3	3	3	3	4	4	6	5
3	3	3	4	3	4	4	6	6	4	3	3	3	4	4	4	4	4	4	3
3	3	3	4	6	6	6	3	4	3	3	3	3	4	3	4	3	3	3	3
3	3	3	3	4	3	6	6	6	4	3	3	3	4	4	4	3	3	3	3
3	3	3	3	4	4	4	6	4	6	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	7	6	7	3	3	3	3	3	4	6	6	4	3
3	3	3	3	3	3	3	6	4	6	4	4	4	3	3	4	4	4	4	3
3	3	3	3	3	3	3	4	4	4	4	3	4	3	3	4	6	6	6	5
3	3	3	3	3	3	4	4	4	3	4	4	4	3	3	3	4	4	6	3
3	3	3	3	4	4	6	3	4	3	3	3	3	3	4	4	6	3	6	5
3	3	3	3	5	3	7	5	5	3	3	3	3	3	5	3	7	5	5	3

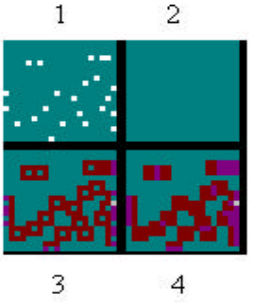
Sur ce type de données le filtre « moyenne » ne donne pas satisfaction, il dénature davantage l'image.

```
Résultats du filtre passe_bas

  3   3   3   5   5   5   3   3   3   3   3   3   3   3   5   5   5   5   5   6
  3   3   4   5   5   4   3   4   5   5   4   3   3   3   4   4   4   4   4   5
  3   3   4   4   4   3   3   4   5   5   4   3   3   3   3   3   3   3   3   3
  3   3   4   4   4   3   3   4   5   7   5   4   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   3   4   4   4   3   3   3   3   3   3   3   3
  3   3   3   3   3   4   4   4   3   4   5   5   5   4   4   3   3   3   3   3
  5   4   4   3   4   5   5   4   3   3   4   5   8   7   5   3   3   3   3   3
  5   4   4   3   5   7   7   4   3   3   4   5   8   7   5   3   3   3   3   3
  5   4   4   3   5   5   5   3   3   3   3   4   5   5   4   3   3   3   3   3
  3   3   3   3   4   4   4   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   4   4   4   3   4   5   5   5   4   4   4   4   4   3   3   3   3   3   3
  5   5   5   4   3   4   5   5   7   5   5   4   4   4   3   4   4   4   3   3
  5   5   5   4   3   4   5   5   7   5   5   4   4   4   3   4   4   4   3   3
  5   4   4   3   3   3   3   3   4   4   4   3   3   3   3   4   4   4   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   4   4   4   3   3   3   3   3   3   3   3   4   4   4   3   3
  3   3   3   3   4   4   4   3   3   3   3   3   3   3   3   4   4   4   3   3
  3   3   3   3   4   4   4   3   3   3   3   3   3   3   3   4   4   5   4   5
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   5   5   6
```

Ici le filtre « passe_bas » a presque le même effet que le filtre « moyenne ».

Affichage des quatre images :



1. Image initiale
2. Effet du filtre médian
3. Effet du filtre "moyenne"
4. Effet du filtre "passe_bas"

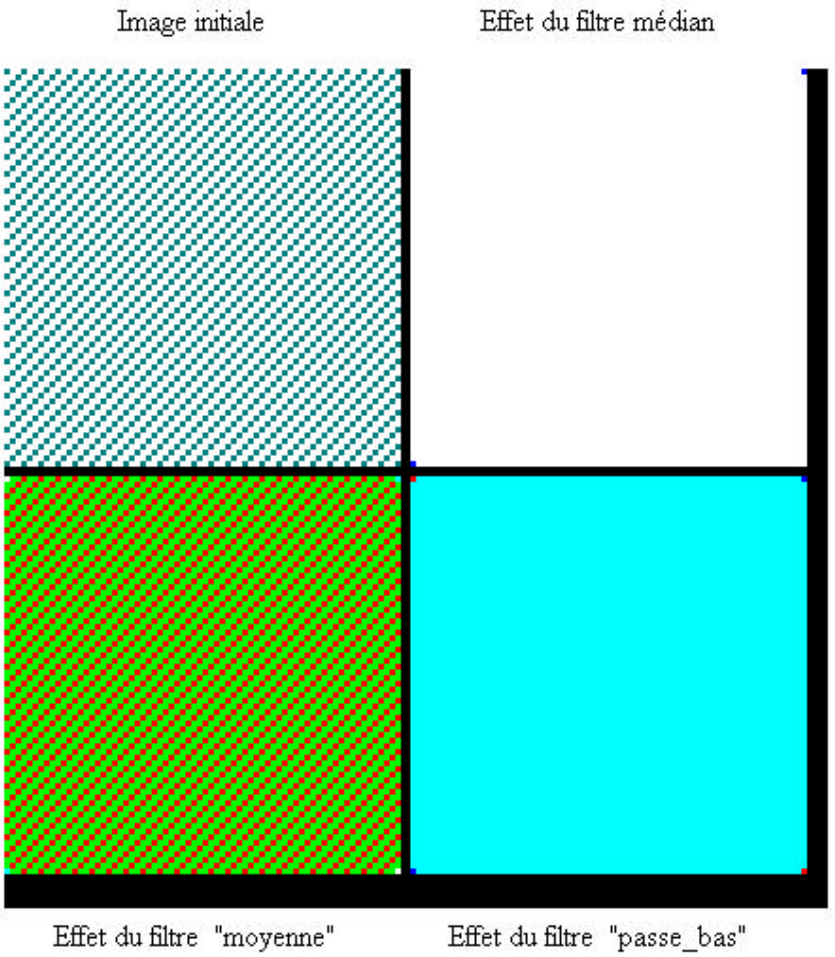
Le filtre médian a rendu l'image nette ; les autres par contre l'ont dégradée.

Test2 :

nb_lig=70, nb_col=70

Dans creation, remplacer la boucle « for » par :

```
for (i=0;i<ligne*colonne;i++)
  {  if(!(i%23)) *(p+i)=4;
     else        *(p+i)=15;
  }
```

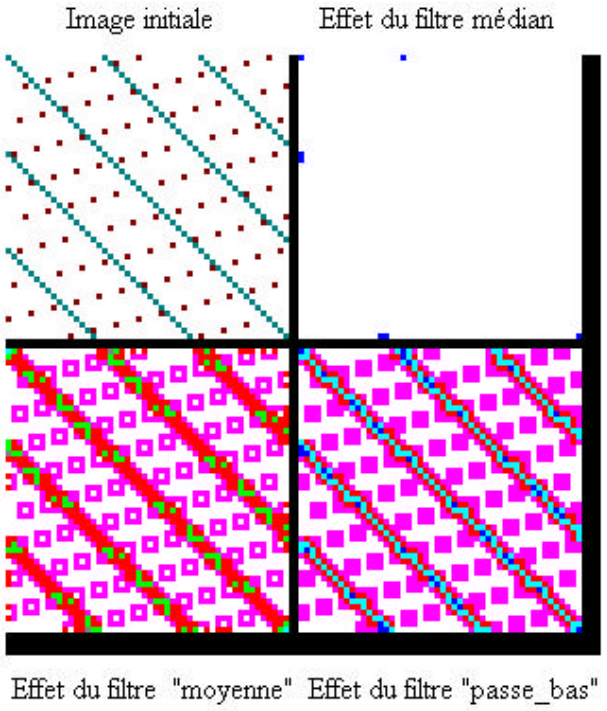


Test3 :

nb_lig=50, nb_col=50.

Dans creation, remplacer la boucle « for » par :

```
for (i=0;i<ligne*colonne;i++)
  {  if(!(i%23)) *(p+i)=4;
        else if(!(i%17)) *(p+i)=3;
             else       *(p+i)=15;
   }
```



CONCLUSION

Après un survol des techniques de traitement d'images, nous avons implémenté et testé trois d'entre elles sur des tableaux de pixels. Des tests sur des images de la région des deux sites pilotes restent à faire, de même que l'implémentation d'autres algorithmes de restauration et d'amélioration d'images.

BIBLIOGRAPHIE

[1] Andrews, H.C. et Hunt B.R.
Digital Image Restoration
Prentice-Hall, Englewood Cliffs, N.J. . 1977

[2] Bellman R.
Introduction to Matrix Analysis
2nd ed., McGraw-Hill, New York. 1970.

[3] B. R. hunt.
The application of constrained least squares estimation to image restoration by digital computer. IEEE Trans. Computer, C-22(9) : 805-812, 1973.

[4] Helstrom C.W.
Image Restoration by the Method of Least Squares.
J. Opt. Soc. Am. vol. 57, no. 3, pp. 297-303. 1967.

[5] Jain A.K. and Angel E.
Image Restoration, Modeling, and Reduction of Dimensionality.
IEEE Trans. Computers., vol. C-23, pp. 470-476. . 1974

[6] Noble B.
Applied Linear Algebra,
Prentice-Hall, Englewood Cliffs, N.J. 1969.

[7] Radu Horaud, Olivier Monga
Vision assistée par ordinateur.Les outils foncdamentaux .Editions Hermès

[8] Philippe Cottet.
1 week end pour comprendre et utiliser l'image et le graphisme.
Osman Eyrolles Multimédia. 1999.

[9] Phillips D.L.
A Technique for the Numerical Solution of Certain Integral Equations of the First Kind.
J. Assoc. Comp. Mach., vol. 9, pp. 84-97. 1962

[10] P. Charbonnier.
Reconstruction d'image : Régularisation avec prise en compte des discontinuités.
Thèse, Université de Nice Sophia-Antipolis, Septembre 1994.

[11] P. Perona and J. Malik.
Scale-space and edge detection using anisotropic difusion.
IEEE Trans. Patern. Analysis and Machine Intelligence, 12(7) : 629-639, 1990.

[12] P. Perona, T. Shiota, and J. Malik.
Anisotropic diffusion
In Bart M. ter Haar Romeny, editor, geometry-driven Diffusion in computer vision, page 7-79.

[13] P. Provent.
Cours de traitement des images.
2ème année .SPE-Cours de traitement d'images.Avril 1995.

[14] Rachid deriche et Olivier faugeras.
Les EDP en Traitement des Images et Vision par Ordinateur.
Traitement du Signal, 13(6), 1996.

[15] Rafael C. Gonzalez, Richard E. Woods.
Digital image processing.
Addison-Wesley, Septembre 1993.

[16] Twomey, S.
On the Numerical Solution of Fredholm Integral Equations of the First Kind by the Inversion of the Linear System Produced by Quadrature
J. Assoc. Comp. Mach., vol. 10, pp. 97-101. 1963