



# ANALYSE D'IMAGE

Rapport de TD5

Guénon Marie et Favreau Jean-Dominique  
VIM / Master SSTIM

## Table des matières

---

Introduction.....	2
Calcul du gradient.....	2
Dédution et valeurs propres.....	3
Comparaison .....	3
Régularisation de Tychonov .....	6
Calcul du gradient.....	6
Observation et comparaison .....	7
Variation de $\lambda$ .....	8
Annexes .....	9
MethVar.m .....	9
divm3.m.....	10

## Introduction

---

Tout d'abord, nous commençons par appliquer une transformation ainsi qu'un terme aléatoire à notre image, et nous obtenons ainsi une image floutée, que l'on peut voir ci-après à gauche comparée avec l'image originale à droite :



A partir de cette image floutée, nous cherchons maintenant à reconstruire l'image originale en utilisant un algorithme de descente de gradient. Pour cela, nous allons mettre en place l'algorithme suivant :

$$\begin{cases} x_{k+1} = x_k - \alpha \nabla J(x) \\ x_0 = \text{léna transformée} \\ \nabla J(x) = 2H^*(Hx - y) \end{cases}$$

## Calcul du gradient

Démontrons déjà que  $\nabla J(x) = 2H^*(Hx - y)$  :

$$\begin{aligned} J(x + h) &= \|H(x + h) - y\|^2 \\ &= \langle H(x + h) - y | H(x + h) - y \rangle \\ &= \langle Hx + Hh - y | Hx + Hh - y \rangle \\ &= \langle Hx - y | Hx + Hh - y \rangle + \langle Hh | Hx + Hh - y \rangle \\ &= \langle Hx - y | Hx - y \rangle + \langle Hx - y | Hh \rangle + \langle Hh | Hh \rangle + \langle Hh | Hx - y \rangle \\ &= \|Hx - y\|^2 + 2\langle Hx - y | Hh \rangle + \|Hh\|^2 \\ &= J(x) + 2\langle H^*(Hx - y) | h \rangle + \|Hh\|^2 \end{aligned}$$

Par identification, nous avons  $\nabla J(x) = 2H^*(Hx - y)$  qui est bien la formule que nous cherchions à démontrer.

Une fois implémenté, cet algorithme nous donne :

```
for k=1:K
    % Calcul du gradient
    gradJ = 2*ifft2(Hetoile.*fft2(ifft2(H.*fft2(xk))-y));
    xk = xk - alpha*gradJ;
    % Calcul de l'erreur et la fonction de cout
    J(k) = norm(ifft2(H.*fft2(xk))-y)^2; % fonction cout
    err(k) = norm(x-xk)^2; % erreur entre xk et l'image originale
    % Affichage
    figure(3); imagesc(xk); colormap gray; title(sprintf('Iteration %d', k)); axis image; axis off;
end
```

Dans notre cas, nous avons pris  $\alpha = 0,5$  et  $K = 1000$ .

## Déduction et valeurs propres

## Comparaison

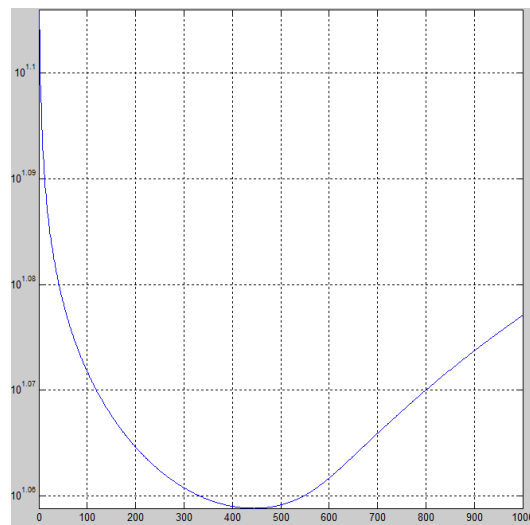
En faisant tourner notre programme de restauration d'image, nous avons constaté une amélioration de l'image transformée dans un premier temps. Comme nous pouvons ici le constater où nous comparons l'image transformée (à gauche) et l'image améliorée après 116 itération (à droite).



Cependant, nous avons aussi constaté qu'après un certain nombre d'itérations, l'image cesse de s'améliorer et nous pouvons voir une nette augmentation du bruit au sein de l'image restaurée. Comme nous pouvons le voir ci-après lors de la comparaison de l'image restaurée après 116 itérations (à gauche) et de l'image restaurée après 1000 itérations (à droite).



Ceci se voit aussi très bien sur la fonction d'estimation d'erreur qui compare la distance entre l'image restaurée au fil des itérations et l'image originale :



Comme nous pouvons le voir, l'erreur commence par diminuer puis après environ 450 itérations augmente nettement. Ceci reflète nettement l'explosion du bruit au cours de la restauration de l'image.

Ce problème de bruit est due au fait que nous n'avons pas de terme de régularisation dans notre algorithme de descente de gradient, et donc accordons de l'importance aux bruits qui finissent par l'emporter sur l'information essentielle de l'image. Pour résoudre ce problème, il faudrait donc imposer

un terme de régularisation dans notre équation  $x_{k+1} = x_k - \alpha \nabla J(x)$ . Pour cela, comme nous le verrons après, nous avons décidé d'utiliser une régularisation de Tychonov d'ordre 1.

## Régularisation de Tychonov

### Calcul du gradient

Le terme de régularisation de Tychonov se retrouve dans  $J(x)$ , qui ne s'écrit plus  $J(x) = \|Hx - y\|^2$  mais  $J(x) = \|Hx - y\|^2 + \lambda \|\nabla x\|^2$ , nous avons donc à recalculer  $\nabla J(x)$  :

$$\begin{aligned}\|\nabla(x+h)\|^2 &= \langle \nabla(x+h) | \nabla(x+h) \rangle \\ &= \langle \nabla x + \nabla h | \nabla x + \nabla h \rangle \\ &= \|\nabla x\|^2 + 2\langle \nabla x | \nabla h \rangle + \|\nabla h\|^2 \\ &= \|\nabla x\|^2 + 2\langle \nabla^*(\nabla x) | h \rangle + \|\nabla h\|^2 \\ &= \|\nabla x\|^2 + 2\langle -\text{div}(\nabla x) | h \rangle + \|\nabla h\|^2\end{aligned}$$

D'où par identification :

$$\nabla(\|\nabla x\|^2) = -2\text{div}(\nabla x)$$

Et en recomposant tous les termes, on obtient :

$$\nabla J(x) = 2H^*(Hx - y) - 2\lambda \text{div}(\nabla x)$$

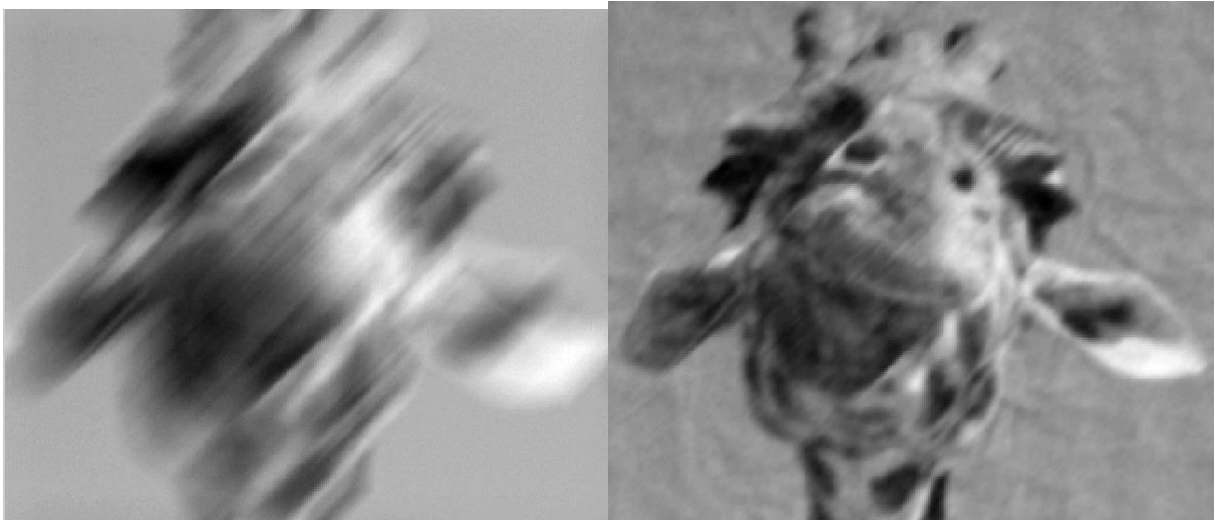
Ce qui nous donne le code suivant :

```
for k=1:K
    % Calcul du gradient
    gradJ = 2*ifft2(Hetoile.*fft2(ifft2(H.*fft2(xk))-y))-2*lambda*divm2(gradm2(xk));
    xk = xk - alpha*gradJ;
    % Calcul de l'erreur et la fonction de cout
    J(k) = norm(ifft2(H.*fft2(xk))-y)^2; % fonction cout
    err(k) = norm(x-xk)^2; % erreur entre xk et l'image originale
    % Affichage
    figure(3);imagesc(xk);colormap gray;title(sprintf('Iteration %d', k)); axis image; axis off;
end
```

Nous appliquons notre algorithme avec les paramètres  $\lambda = 0,01$  ;  $\alpha = \frac{1}{2(1+8\lambda)}$  et  $K = 1000$ .

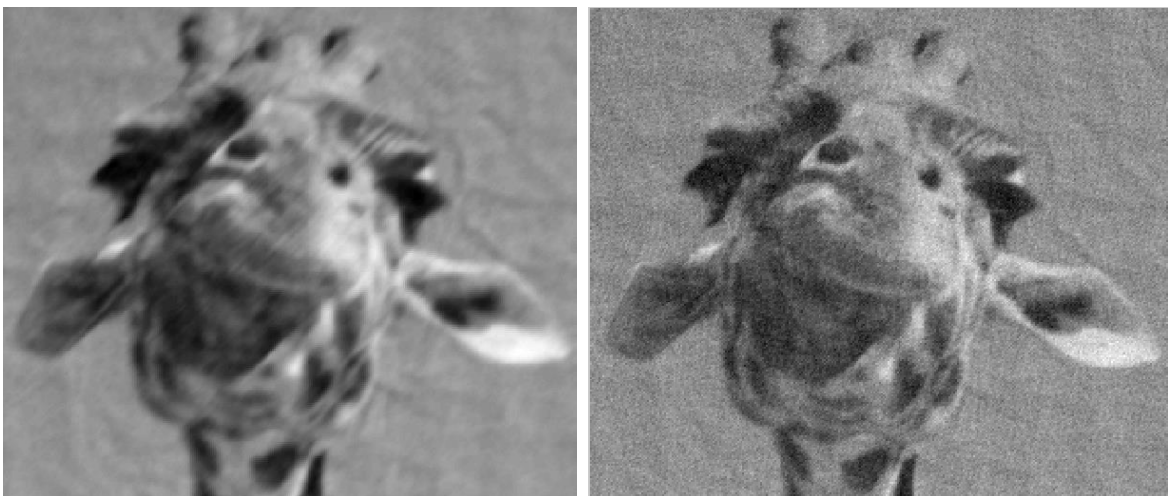
## Observation et comparaison

En appliquant l'algorithme de descente de gradient ainsi obtenue sur une image dégradée (à gauche), nous pouvons voir apparaître une image restaurée (à droite) à la qualité tout à fait acceptable et où l'on reconnaît une girafe.



Si l'on compare les résultats obtenus sur cette image avec (à gauche) et sans (à droite) régularisation, nous pouvons constater que dans le premier cas, nous obtenons une image floutée. Ceci est due au fait que nous effectuons une sorte de lissage sur l'image pour enlever les effets de bruit.



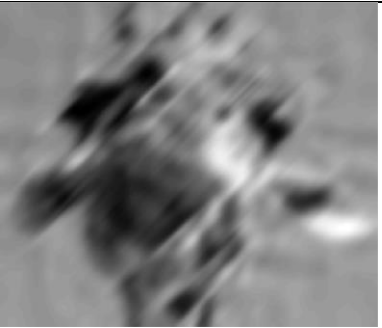
Cependant, sans régularisation nous avons une image plus nette si l'on fait abstraction du bruit. Mais d'une part la stabilité de l'algorithme n'est pas assurée et d'autre part, si l'on augmente trop le nombre d'itération, on se rapproche de la solution exacte donnée par Fourier qui est totalement illisible.





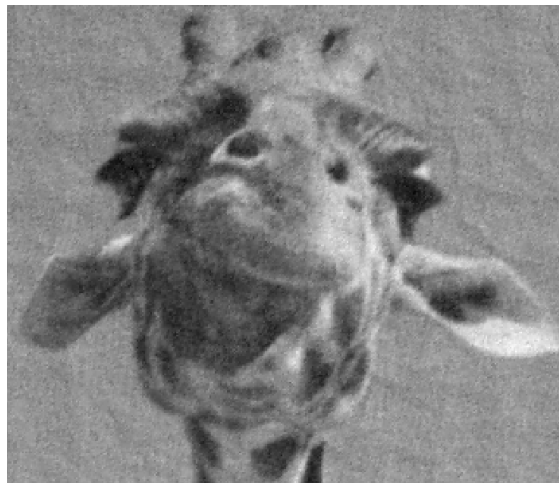
Pour éviter le flou due à la régularisation, nous pourrions utiliser une méthode anisotropique, comme vue lors du TP précédent, qui effectuerait le lissage évoqué ci-dessus seulement aux endroits où il n'y a pas de contours et éviterait ainsi de trop flouter l'image.

## Variation de $\lambda$

$\lambda$	0.001	0.01	0.5
Image restaurée			

Nous pouvons constater que la variation de  $\lambda$  influence grandement la qualité de l'image restaurée et plus particulièrement sa netteté. En effet, nous pouvons constater que plus  $\lambda$  est petit ( $\lambda=0.001$ ), moins d'importance est donnée au terme de régularisation, ce qui donne une image plus bruitée. A contrario, plus  $\lambda$  est grand ( $\lambda=0.5$ ), plus on donne d'importance au terme de régularisation et donc plus on se retrouve avec une image floue en sortie.

C'est pourquoi il faut savoir prendre un  $\lambda$  raisonné entre ces deux extrêmes pour éviter ces désagréments. Ou bien, comme il a déjà été dit plus haut, nous pouvons aussi faire appel à une méthode anisotropique qui permettrait de lisser l'image seulement aux endroits où il n'y a pas de contours et donc ainsi d'éviter le flou, ce qui nous donnerait un résultat du type :



## Annexes

---

### MethVar.m

```
clear all; close all; clc;

% -- Parametres
lambda = 0.01; % Ponderation de la regularisation de
Tychonov d'ordre 1
alpha = 1/(2*(1+8*lambda)); % Pas de la descente de
gradient
K = 1000; % Nombre d'iterations de la descente de
gradient
sig2 = 1; % Variance du bruit additif Gaussien

% -- Lecture de l'image
x = double(imread('lena.bmp')); % Chargement de l'image reelle x

% -- Construction des operateurs H et H*
% H sera un filtre passe-bas Gaussien de variance 1
h = fspecial('gaussian', 11, 1);

s = (size(h)-1)/2;

Hs = zeros(size(x));
Hs(1:s(1)+1,1:s(2)+1) = h(s(1)+1:end,s(2)+1:end);
Hs(end+1-s(1):end,end+1-s(2):end) = h(1:s(1),1:s(2));
Hs(1:s(1)+1,end+1-s(2):end) = h(s(1)+1:end,1:s(2));
Hs(end+1-s(1):end,1:s(2)+1) = h(1:s(1),s(2)+1:end);

%==== Pour l'image à la fin du TP =====
load('Hs_motion_blur');
x=double(imread('motion_blur.png'));
%=====

H = fft2(Hs); % H, Hx se calculera par ifft2(H.*fft2(x));
Hetoile = conj(H); % H*, H*x se calculera par ifft2(Hetoile.*fft2(x));

% -- Construction de l'image observee y
y = x;%ifft2(H.*fft2(x)) + sqrt(sig2).*randn(size(x));
% Affichage de l'image observee y
figure(1);imagesc(y);colormap gray;title('Image observee'); axis image;
axis off;

% -- Calcul de la solution exacte par Fourier
xsol = ifft2(Hetoile.*fft2(y)./(abs(H).^2));
figure(2);imagesc(xsol);colormap gray; title('Solution exacte par
Fourier'); axis image; axis off;

% -- Descente de gradient
% Initialisation
xk = y;
J = zeros(1, K); % Fonction de cout
```

```
err = zeros(1, K); %Fonction d'erreur
% Boucle principale
for k=1:K
    % Calcul du gradient
    gradJ = 2*ifft2(Hetoile.*fft2(ifft2(H.*fft2(xk))-y))-
2*lambda*divm2(gradm2(xk));
    xk = xk - alpha*gradJ;
    % Calcul de l'erreur et la fonction de cout
    J(k) = norm(ifft2(H.*fft2(xk))-y)^2; % fonction cout
    err(k) = norm(x-xk)^2; % erreur entre xk et l'image originale
    % Affichage
    figure(3);imagesc(xk);colormap gray;title(sprintf('Iteration %d', k));
axis image; axis off;
end

% -- Affichages
% Image restauree
figure(3);imagesc(xk);colormap gray;title('Image restauree'); axis image;
axis off;
% Image originale
figure(4);imagesc(x);colormap gray;title('Image originale'); axis image;
axis off;
% Erreur d'estimation
figure;semilogy(log(err));title('Erreur d'estimation'); grid;
% Fonction cout
figure;plot(J);title('Evolution de la fonction cout');grid;
```

## divm3.m

cette fonction sert à calculer la divergence dans le cas anisotropique

```
%function im=divm(gim)
%divergence CHambolle scheme for B and W images
function im=divm2(gim)

    function cs=c(s)
        cs=1./sqrt(1+s);
    end
    function cs=cprim(s)
        cs=-1./(2*(sqrt(1+s)).^3);
    end

    [ny,nx,m]=size(gim);

    ux = zeros(ny,nx);
    ux(:,1:end-1) = gim(:,2:end,1) - gim(:,1:end-1,1);

    uy = zeros(ny,nx);
    uy(1:end-1,:) = gim(2:end,:,2) - gim(1:end-1,:,2);

    norm_grad2 = ux.*ux + uy.*uy;
    cgrad = c(norm_grad2);
    cprim_grad = cprim(norm_grad2);

    uxx=zeros(ny,nx);
    uxx(:,1)=gim(:,1,1);
    uxx(:,end)=-gim(:,end-1,1);
```

```
uxx(:,2:end-1)=gim(:,2:end-1,1)-gim(:,1:end-2,1);

uyy=zeros(ny,nx);
uyy(1,:)=gim(1,:,2);
uyy(end,:)=gim(end-1,:,2);
uyy(2:end-1,:)=gim(2:end-1,:,2)-gim(1:end-2,:,2);

uxy=zeros(ny,nx);
uxy(:,1)=gim(:,1,2);
uxy(:,end)=gim(:,end-1,2);
uxy(:,2:end-1)=gim(:,2:end-1,2)-gim(:,1:end-2,2);

uee=(ux.*ux).*uxx + (uy.*uy).*uyy + 2.*ux.*uy.*uxy;%./norm_grad2;
unn=(ux.*ux).*uyy + (uy.*uy).*uxx - 2.*ux.*uy.*uxy;%./norm_grad2;
for row = 1 : size(uee,1)
    for col = 1 : size(uee,2)
        aa = norm_grad2(row,col);
        if(aa~=0)
            uee(row,col) = uee(row,col)/aa;
            unn(row,col) = unn(row,col)/aa;
        end
    end
end

im=cgrad.*uee+(cgrad+2*norm_grad2.*cprim_grad).*unn;
end
```