PALLAS Léo
REINERT Pauline

4 ETI
Groupe B

Traitement et Synthèse d'Image
TP1 – Filtrage

I. Filtrage passe-haut dans l'espace direct : détection de contours

On souhaite détecter les contours de cellules à l'aide du filtre de Sobel, défini par les masques :

$$M_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ et } M_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Le filtre de Sobel se décompose en produit matriciel du filtre dérivatif 1D $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$ et du filtre de lissage 1D $\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$.

On utilise une fonction de lissage à la fois pour obtenir une fonction continue différenciable et pour réduire le bruit, qui peut fausser la détection.

On se propose d'effectuer une détection de contour sur l'image 'flower.png'. Pour cela, on calcule les composantes horizontales et verticales du gradient en chaque pixel de l'image I à l'aide du filtre de Sobel, ce qui nous permet de déduire la norme du gradient. En effet, $\|\vec{G}\| = \sqrt{{G_x}^2 + {G_y}^2}$, avec $G_x(i,j) = M_1 * I(i,j)$ et $G_y(i,j) = M_2 * I(i,j)$.

Comme un contour est détecté par un maximum de la dérivée première de l'image dans la direction du gradient, les maxima de la norme du gradient correspondent à la présence d'un contour.

Code Matlab développé :

```
A=imread('flower.png'); %lecture de l'image
figure(1);
subplot(221);
imshow(A); %Affichage de l'image
title('Image originale');
B=im2double(A); %Conversion en précision double
```

```
[h,w]=size(B); %Hauteur et largeur de l'image
filtreh=[-1,0,1;-2,0,2;-1,0,1]; %Définition du filtre de Sobel horizontal
filtrev=[-1,-2,-1;0,0,0;1,2,1]; %Définition du filtre de Sobel vertical
Gh=imfilter(B,filtreh); %Filtrage de l'image par le filtre horizontal
subplot(223)
imshow(Gh,[min(Gh(:)) max(Gh(:))]);
title('Composante horizontale du gradient');
Gv=imfilter(B,filtrev); %Filtrage de l'image par le filtre vertical
subplot(224);
imshow(Gv,[min(Gv(:)) max(Gv(:))]);
title('Composante verticale du gradient');
G=sqrt((Gh.^2)+(Gv.^2)); %Calcul de la norme du gradient
subplot(222);
imshow(1-G,[min(1-G(:)) max(1-G(:))]);
title('Norme du gradient');
```

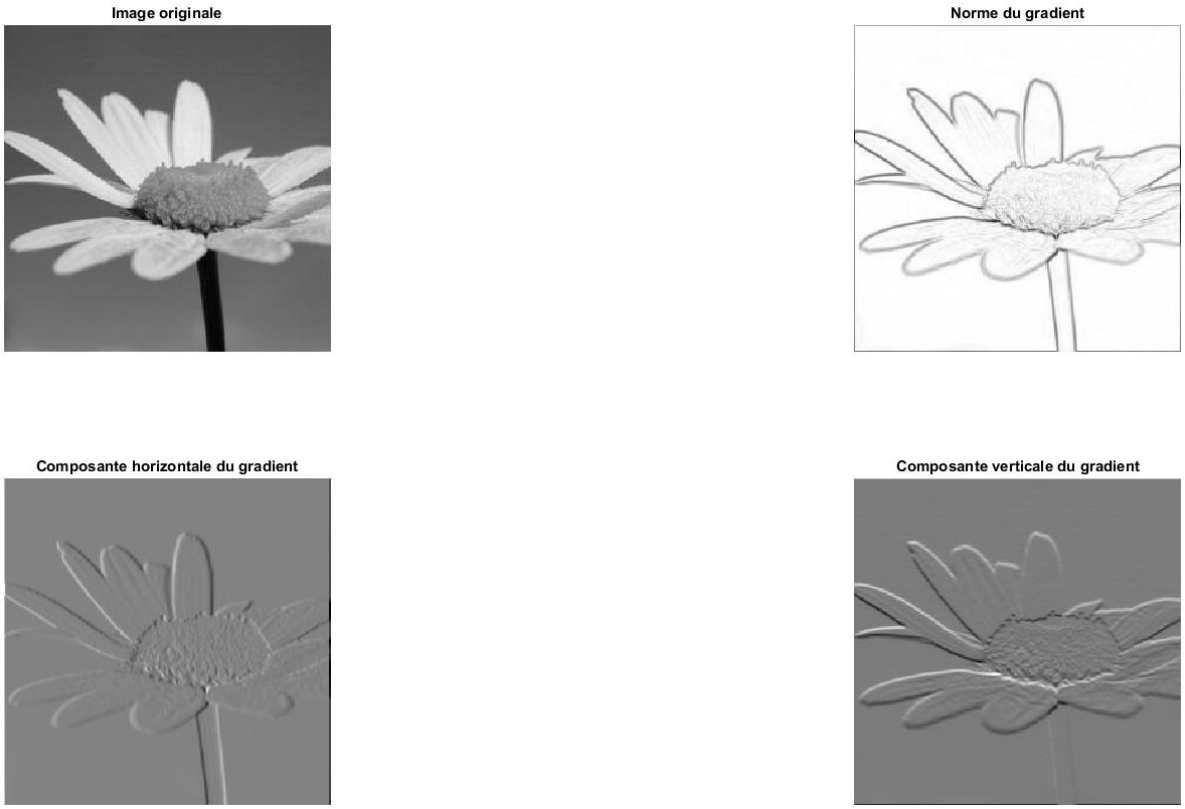

Figure 1 : Détection de contour sur l'image flower.png

On rajoute maintenant un bruit gaussien centré de variance 0.01 à l'image de départ. On reprend la méthode précédente pour effectuer une détection de contour.

Code Matlab développé :

```
B2=imnoise(B,'gaussian',0.01); %Introduction d'un bruit gaussien dans l'image
figure(2);
subplot(221);
imshow(B2);
title('Image avec bruit gaussien');
Gh2=imfilter(B2,filtreh); %Filtrage de l'image par le filtre horizontal
subplot(223)
imshow(Gh2,[min(Gh2(:)) max(Gh2(:))]);
title('Composante horizontale du gradient');
Gv2=imfilter(B2,filtrev); %Filtrage de l'image par le filtre horizontal
subplot(224);
imshow(Gv2,[min(Gv2(:)) max(Gv2(:))]);
title('Composante verticale du gradient');
G2=sqrt((Gh2.^2)+(Gv2.^2)); %Calcul de la norme du gradient
subplot(222);
imshow(1-G2,[min(1-G2(:)) max(1-G2(:))]);
title('Norme du gradient');
```

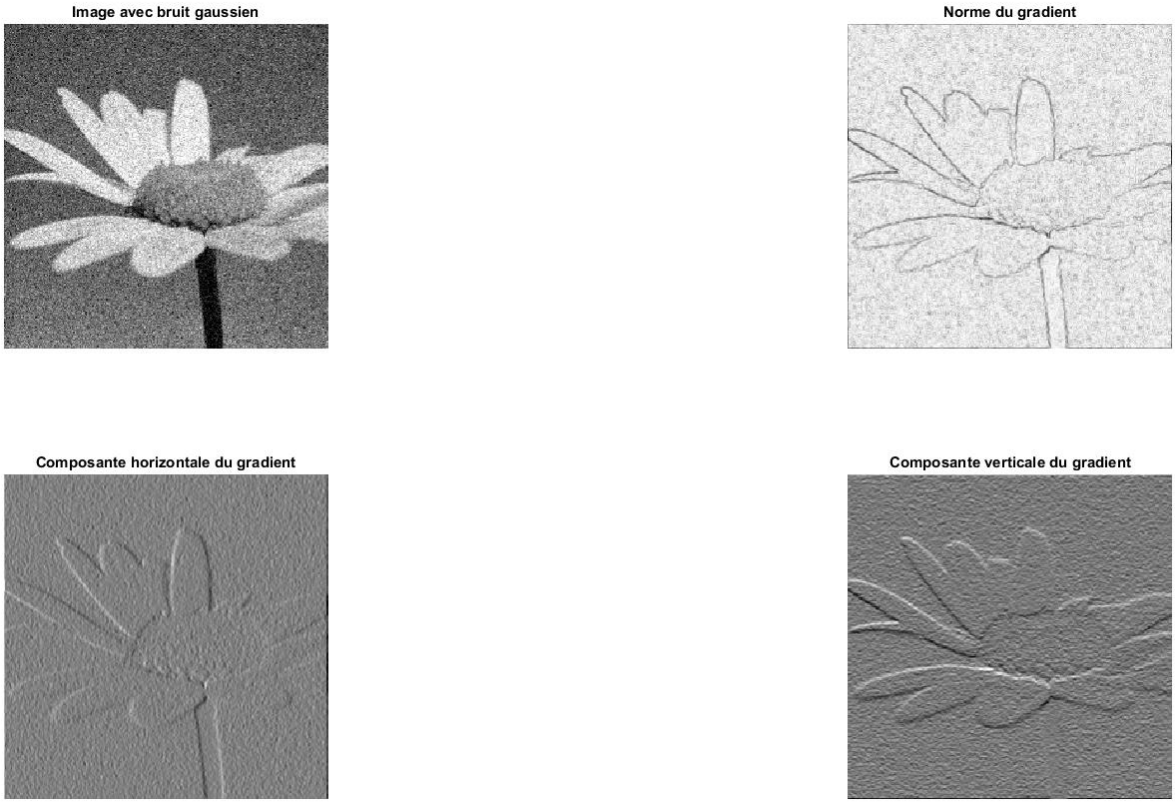

Figure 2 : Affichage de la norme du gradient de l'image bruitée

On remarque que le bruit se répercute beaucoup sur la norme du gradient, on ne peut pas l'exploiter directement comme une détection de contour, comme on pouvait le faire sur la figure 1.

Un contour correspond à un maximum local dans la direction du gradient. Ainsi, pour chaque pixel p de l'image, on cherche les coordonnées des deux points p_1 et p_2 situés sur la droite de vecteur

directeur le gradiant $\vec{G}$, qui sont à une distance $d = 2$ pixels du pixel p. On considère alors que le pixel p appartient à un contour si $\|\vec{G}(p)\| > \|\vec{G}(p_1)\|$ et $\|\vec{G}(p)\| > \|\vec{G}(p_2)\|$. Cette expression se traduit par le fait qu'on considère qu'un pixel est un contour si le gradient en ce pixel est supérieur aux gradients des pixels situés à une distance $d = 2$ pixels de lui dans la direction du gradient.

Comme le bruit forme une nappe aléatoire de gradients additionnée au gradient de l'image, il y a une probabilité assez faible (mais non nulle) qu'un pixel bruité ait un gradient supérieur aux gradients des pixels situés localement autour de lui dans la direction de son gradient, s'il ne fait pas partie d'un contour de l'image originale.

Code Matlab développé :

```
Gh_n=Gh2./(sqrt(Gh2.^2+Gv2.^2)); %Normalisation de la composante horizontale du
gradient
Gv_n=Gv2./(sqrt(Gh2.^2+Gv2.^2)); %Normalisation de la composante verticale du
gradient
d=2; %distance au pixel choisie
[X,Y]=meshgrid(1:w,1:h); %Création des coordonnées X et Y des pixels
p1X=X+d.*Gh_n; %Calcul de la composante horizontale de p1
p1Y=Y+d.*Gv_n; %Calcul de la composante verticale de p1
p2X=X-d.*Gh_n; %Calcul de la composante horizontale de p2
p2Y=Y-d.*Gv_n; %Calcul de la composante verticale de p2
C=zeros(h,w); %Matrice de détection de contour
eps=0.5; %Seuil de détection
p1X(p1X<1)=1; %Correction des indices inférieurs à 1
p2X(p2X<1)=1;
p1Y(p1Y<1)=1;
p2Y(p2Y<1)=1;
p1X(p1X>w)=w; %Correction des indices supérieurs à la taille de l'image
p2X(p2X>w)=w;
p1Y(p1Y>h)=h;
p2Y(p2Y>h)=h;
for i=1:w
   for j=1:h
       if (G2(j,i)-G2(round(p1X(j,i)),round(p1Y(j,i)))>eps)&&(G2(j,i)-
G2(round(p2X(j,i)),round(p2Y(j,i)))>0.5)
           C(j,i)=G2(j,i); %Si la condition du contour est vérifiée, le pixel prend
la valeur de la norme du gradient
           %Sinon le pixel reste à 0
       end
   end
end

figure(3);
subplot(131)
imshow(B2);
title('Image avec bruit gaussien');
subplot(132)
imshow(1-G2,[min(1-G2(:)) max(1-G2(:))]);
title('Norme du gradient de l''image bruitée');
subplot(133)
imshow(1-C,[min(1-C(:)) max(1-C(:))]);
```

```
title('Contours de l''image bruitée');
```

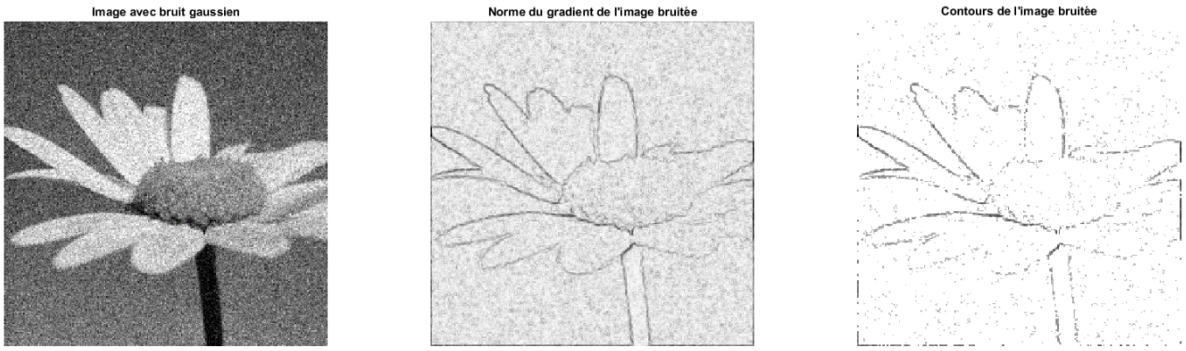

Figure 3 : Détection de contour sur l'image bruitée

On remarque que des contours sont détectés à tort à certains endroits, et que certains contours ne sont pas détectés alors qu'ils devraient l'être à d'autres endroits. Cela est dû au bruit qui fausse l'intensité du gradient, ce qui fait que pour certains cas la condition de détection citée précédemment n'a pas la valeur de vérité qu'elle devrait avoir.

II. Filtrage passe-haut dans l'espace de Fourier

On souhaite corriger le biais d'une photo. En effet, il y a un nuage sur l'image qui la rend floue. On effectue donc un filtrage passe-haut car le nuage correspond aux basses fréquences. On utilise pour cela un filtre de Butterworth.

Code Matlab qui permet de lire, afficher et récupérer la taille de l'image :

```
A = imread('ngc2175.png'); %lit l'image
A2 = im2double(A); %convertit l'image en precision double
figure(1)
imshow(A2); %affiche l'image
title('Image originale')
[h,w] = size(A); %récupère la taille de l'image
```

Figure 2 : Image originale

Code Matlab permettant de générer le filtre de Butterworth :

```
n = 2; %ordre du filtre de Butterworth
fc = 3; %fréquence de coupure
[U,V] = meshgrid(-w/2+1/2:w/2-1/2,-h/2+1/2:h/2-1/2); %matrices donnant les valeurs
u et v des pixels de sorte à ce que le pixel au centre de l'image ait (0,0)  pour
coordonnées
D = zeros(450,600); %initialisation d'une matrice
for k=1:600
    for j=1:450
        D(j,k) = sqrt(U(j,k)^2+V(j,k)^2); %on calcule la distance au centre (norme)
    end
end
H = 1./(1+((fc./D).^(2*n))); %on calcule le filtre de Butterworth
figure(2)
mesh(H) %on affiche le filtre en 3D
title('Représentation 3D du filtre')
```

Le filtre de Butterwoth passe-haut est : $\frac{1}{1+(\frac{fc}{\sqrt{u^2+v^2}})^{2n}}$. Le code précédent nous permet de récupérer les valeurs des abscisses et ordonnées de chaque pixel dans les matrice U et V. On peut ensuite en déduire la matrice de distance au centre de chaque pixel, D, qu'on calcule car c'est une norme : $D = \sqrt{U^2 + V^2}$.

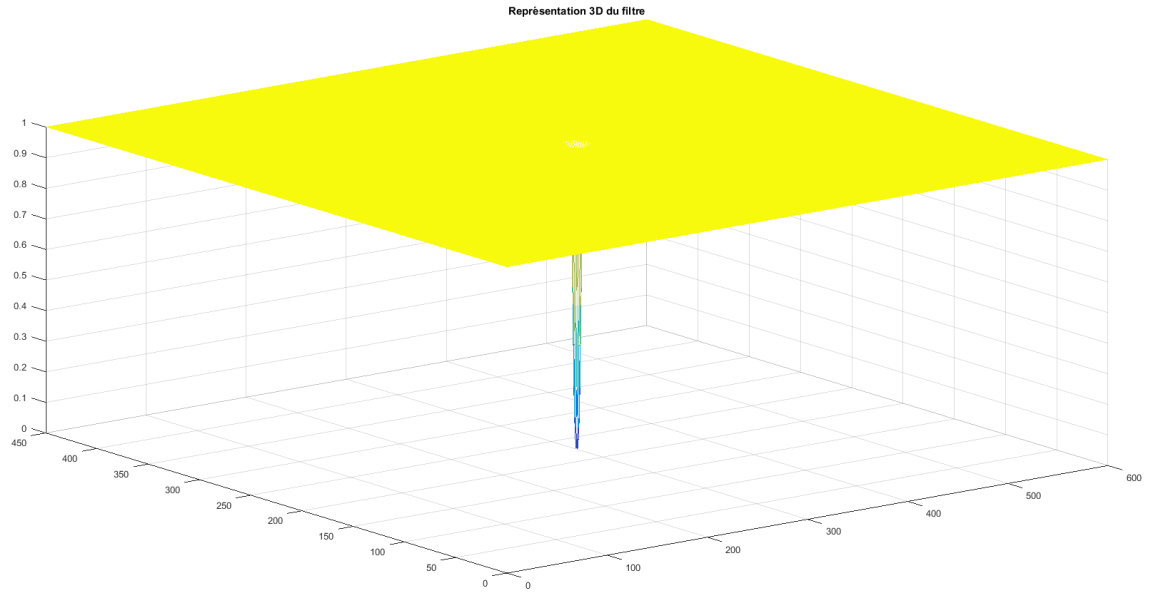

Figure 3 : Représentation 3D du filtre

Code Matlab permettant de calculer la transformée de Fourier de l'image, de la filtrer et de calculer sa transformée de Fourier inverse :

```
TFA = fft2(A2); %calcule la transformée de Fourier de l'image
TFA = fftshift(TFA); %arrange les données pour que la fréquence nulle soit au
centre de l'image

figure(3)
imshow(10*log(abs(TFA)),[]); %affiche le module de la transformée de Fourier de
l'image
title('Module de la transformée de Fourier de l''image')
IMFF = H.*TFA; %on applique le filtre à l'image
IMFF = fftshift(IMFF); %arrange les données pour que la fréquence nulle soit au
centre de l'image
```

On voit que le filtre laisse bien passer les hautes fréquences et coupe rapidement les basses fréquences. En effet, lorsque U et V sont proches de 0 (du centre), le filtre tend vers 0 aussi, et inversement.

Figure 4 : Représentation du module de la transformée de Fourier de l'image

La figure 4 permet de voir que les composantes à éliminer se trouvent dans les très basses fréquences. C'est pour cela qu'on crée le filtre précédent. On filtre l'image en réalisant une convolution de l'image par le filtre dans le domaine direct ce qui revient à multiplier l'image par le filtre dans le domaine fréquentiel.

Code Matlab permettant de revenir dans l'espace direct et d'afficher l'image filtrée :

```
IMF = ifft2(IMFF); %calcule la transformée de Fourier inverse
figure(4)
imshow(IMF,[min(H(:)) max(H(:))]) %affiche l'image filtrée
title('Image filtrée')
```

Figure 5 : Image filtrée par un filtre de Butterworth passe-haut

On constate que le « brouillard » qu'on observait sur l'image originale a disparu mais les étoiles sont aussi un petit peu moins lumineuses. Il reste aussi une petite trace du nuage à droite car sur la figure 4, on peut voir qu'il y a quelques composantes à effacer se trouvant dans les hautes fréquences.

III. Filtrage non linéaire : filtre médian

Un filtre médian récupère les valeurs des pixels compris dans le masque, les classe par ordre croissant, prend la valeur médiane et la met au milieu du masque.

Un filtre d'ordre fait la même chose mais il ne retourne pas la valeur médiane, il retourne la p-ième valeur.

Un bruit poivre et sel est un bruit qui met un pixel soit à 0 soit à 255 de manière aléatoire. Il est logique d'utiliser un filtre médian, qui utilise les pixels voisins afin de retrouver la valeur correcte du pixel concerné.

Code Matlab permettant de lire et d'afficher l'image puis de rajouter un bruit sel et poivre dessus :

```
A = imread('flower.png'); %lire l'image
A2 = im2double(A);%convertit l'image pour avoir une précision double
figure(1)
imshow(A2); %affiche l'image
title('image originale')
B = imnoise(A,'salt & pepper',0.5); %ajout d'un bruit sel et poivre
figure(2)
```

```
imshow(B) %affiche l'image bruitée
title('image bruitée'
```

Figure 7 : Image originale

Figure 6 : Image bruitée par un bruit sel et poivre

On distingue toujours la forme de l'image et un petit peu ses contours.

Code Matlab de correction du bruit à l'aide d'un filtre média, pour différente valeurs de masques :

```
figure(3)
subplot(221)
Bmed = ordfilt2(B,41,ones(9,9));
imshow(Bmed);
title('Masque taille 81');
subplot(222)
Bmed2 = ordfilt2(B,5,ones(3,3));
imshow(Bmed2);
title('Masque taille 9')
subplot(223)
Bmed3 = ordfilt2(B,13,ones(5,5));
imshow(Bmed3)
title('Masque taille 25');
subplot(224)
Bmed4 = ordfilt2(B,25,ones(7,7));
imshow(Bmed4);
title('Masque taille 49');
```

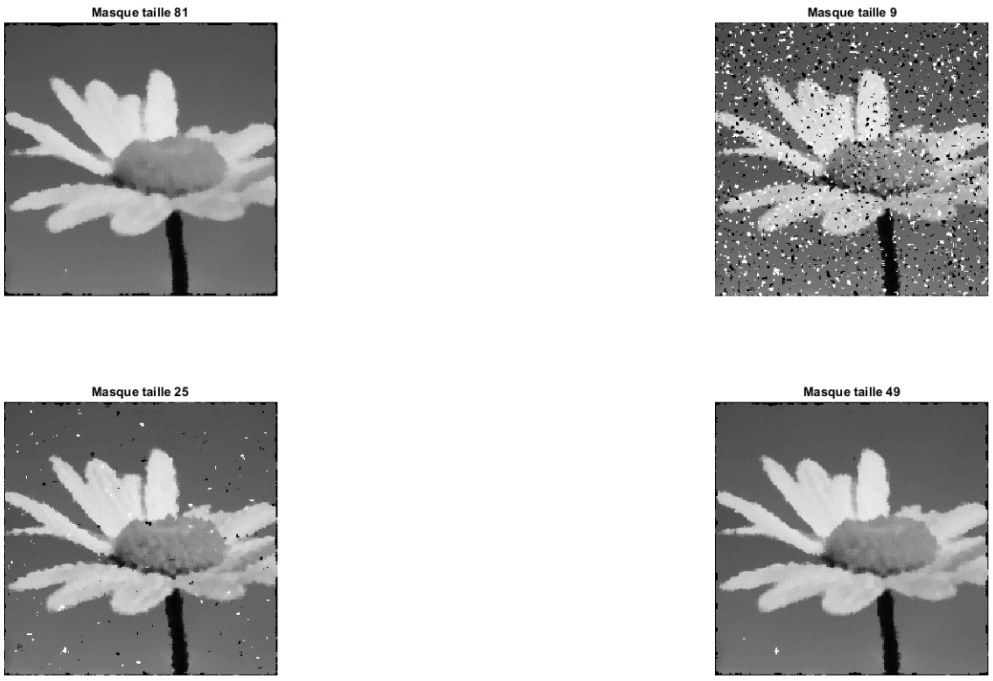

Figure 8 : Image débruitée par un filtre médian pour différentes valeurs de masques

L'image étant très bruitée au départ, il sera forcément impossible d'obtenir une belle image en filtrant. Néanmoins, on peut voir qu'il est possible de retrouver les contours, les couleurs et un peu de texture de l'image.

Avec une image aussi bruitée, un petit masque permet de réduire l'effet de sel et poivre mais celui-ci est toujours présent. En augmentant la taille du masque, on peut ainsi enlever ce bruit. Cependant, en affectant la valeur moyenne des pixels voisins au pixel concerné, on effectue un lissage, et donc on perd les détails de l'image originale.

IV. Filtrage à partir d'histogramme : seuillage par K-means

On réalise un seuillage à deux régions. Pour cela on choisit deux intensités $m1$ et $m2$ aléatoirement entre 0 et 255. Puis pour chaque pixel, on affecte le label 1 si la valeur du pixel est plus proche de $m1$ ou le label 2 si la valeur du pixel est plus proche de $m2$. On actualise ensuite la valeur de $m1$ en prenant la moyenne des intensités des pixels ayant le label 1 et on réalise de même pour $m2$. On réalise ensuite le même processus jusqu'à ce que les valeurs de $m1$ et de $m2$ ne changent plus. On peut alors définir un seuil à la moyenne entre m1 et m2, ce qui nous permet de seuiller dynamiquement l'image. Pour effectuer un seuillage par K-means, il suffit de répéter k fois cette opération sur k segments d'intensité. La méthode des *k*-means est un outil de classification qui permet de répartir un ensemble de données en k classes homogènes. Elle permet de regrouper des objets possédant des propriétés similaires.

Code Matlab permettant de réaliser le seuillage d'une image :

```
A=imread('flower.png'); %lecture de l'image
```

```
[h,w]=size(A);
figure(1);
imshow(A); %Affichage de l'image
title('Image originale');
m1=floor(256*rand) %Choix aléatoire de l'intensité m1 comprise entre 0 et 255
m2=floor(256*rand) %Choix aléatoire de l'intensité m1 comprise entre 0 et 255
labels=zeros(size(A)); %Matrice des labels des pixels
m1p=-1; %intensité m1 précédente (initialisée à -1)
m2p=-1; %intensité m2 précédente (initialisée à -1)
while(m1p~=m1)&&(m2p~=m2)  %Jusqu'à ce que m1 et m2 ne changent plus
    for i=1:h %Pour chaque pixel de l'image
        for j=1:w
            if abs(A(j,i)-m1)<abs(A(j,i)-m2) %Si son intensité est plus proche de
m1 que de m2
                labels(j,i)=1; %On lui affecte le label 1
            else
                labels(j,i)=2; %Sinon on lui affecte le label 2
            end
        end
    end
    m1p=m1; %On stocke m1 dans m1p
    m1=round(mean(A(labels==1))) %m1 prend la valeur de la moyenne des intensités
des pixels de label 1
    m2p=m2; %On stocke m2 dans m2p
    m2=round(mean(A(labels==2))) %m2 prend la valeur de la moyenne des intensités
des pixels de label 2
end

seuil=abs(m1-m2)/2; %On place le seuil au centre du segment [m1;m2]
A((255-A)<seuil)=255; %Si l'intensité d'un pixel est en dessous du seuil, le pixel
est blanc
A((255-A)>=seuil)=0; %Si l'intensité du pixel est au dessus du seuil, le pixel est
noir
figure(2);
imshow(A,[]);
title('Image seuillée')
```

Figure 9 : Image filtrée par seuillage