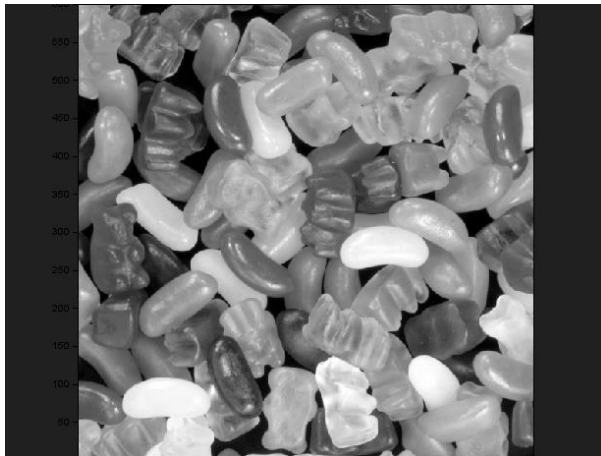


# TP4\_MAP201

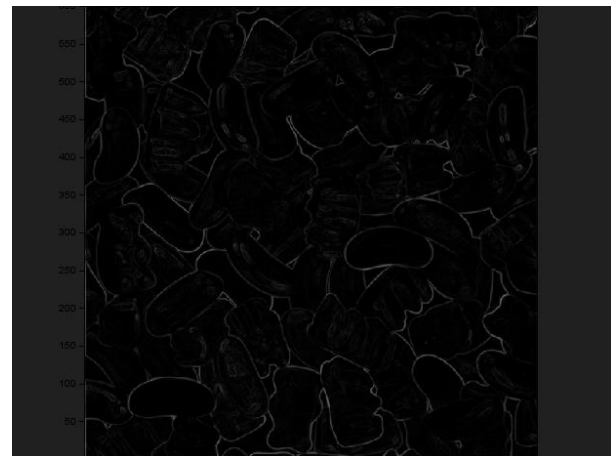
Groupe: INF2

Shaghayagh HAJMOHAMMADKASHI

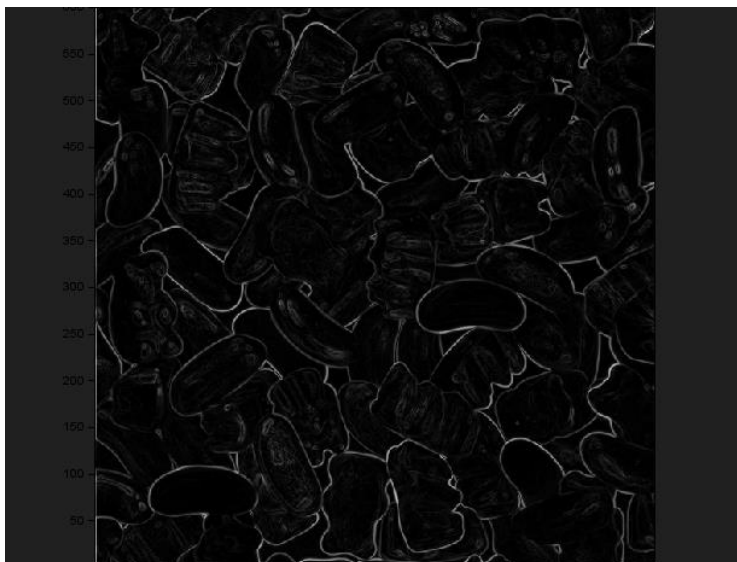
#Exo1



*Image originale*



*l'image de la norme du gradient de l'image originale*

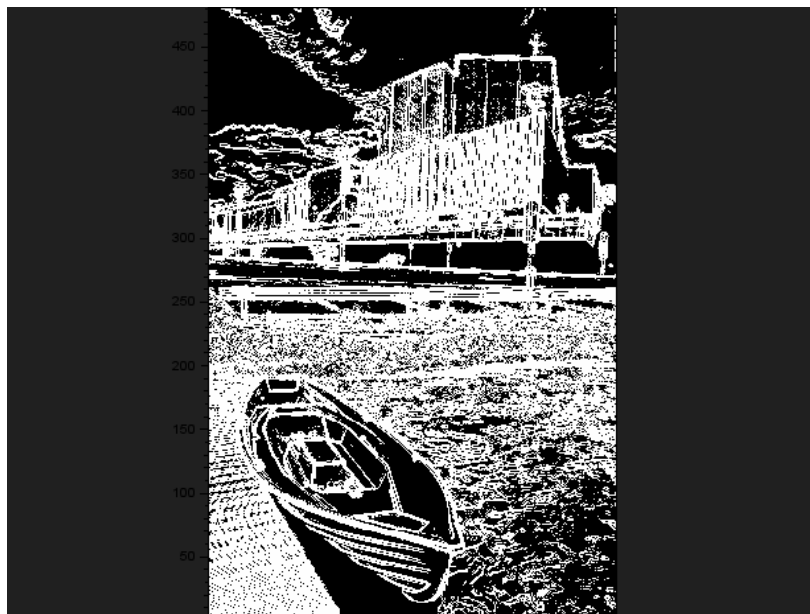


*Après correction de la contraste*

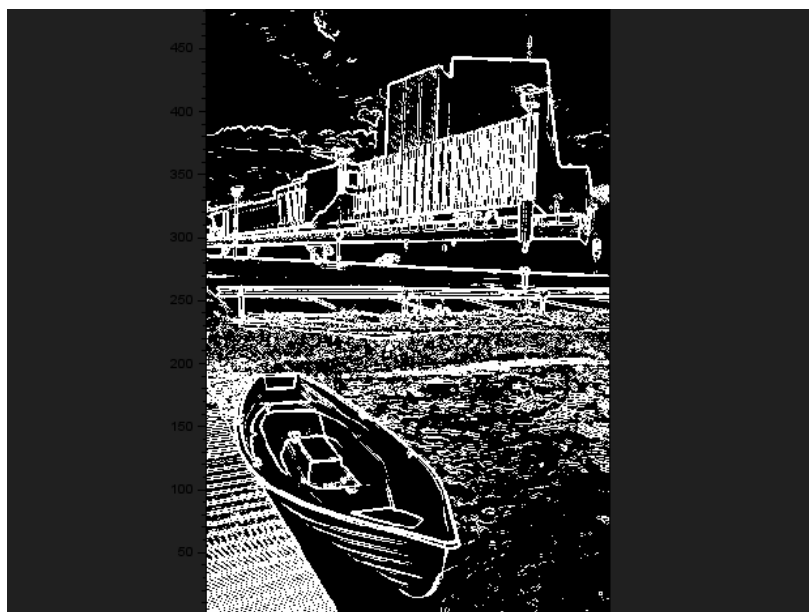
```
imn = (imn - min(imn(:))) * 255 / (max(imn(:)) - min(imn(:)));
```

Cette ligne de code normalise les valeurs de l'image imn entre 0 et 255. La normalisation est effectuée en soustrayant la valeur minimale de l'image imn et en multipliant par 255 divisé par la plage dynamique des valeurs de imn (la différence entre les valeurs maximale et minimale de imn). Cela assure que les valeurs de l'image imn sont équitablement distribuées entre 0 et 255, ce qui est utile pour l'affichage et l'analyse ultérieure de l'image.

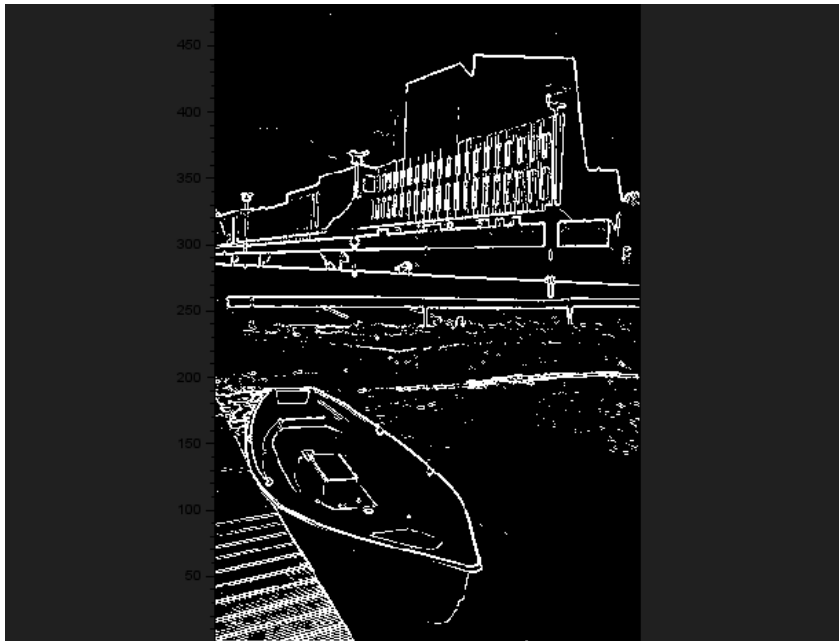
#Exo2



*Seuil 10*



*Seuil 20*



Seuil 50

L'application d'un seuil peut avoir une influence significative sur le résultat final. Si le seuil est trop bas, alors beaucoup de pixels auront une valeur supérieure au seuil, ce qui donnera une image très claire et peu contrastée. À l'inverse, si le seuil est trop élevé, beaucoup de pixels auront une valeur inférieure au seuil, ce qui donnera une image très sombre et peu contrastée. Le choix du seuil dépend donc du contenu de l'image et de l'objectif du traitement.

### #Exo3

```
exec('init_tp_image.sce');
exec('contours.sci', -1);
p = 0.8;
im = lire_imageBMPgrise('barque.bmp');
seuil = trouver_seuil(im, p);
disp(seuil);
//pour p=0,8 seuil=31
//pour p=0,5 seuil=11
//pour p=0 seuil=256
//pour p=1 seuil=1
```

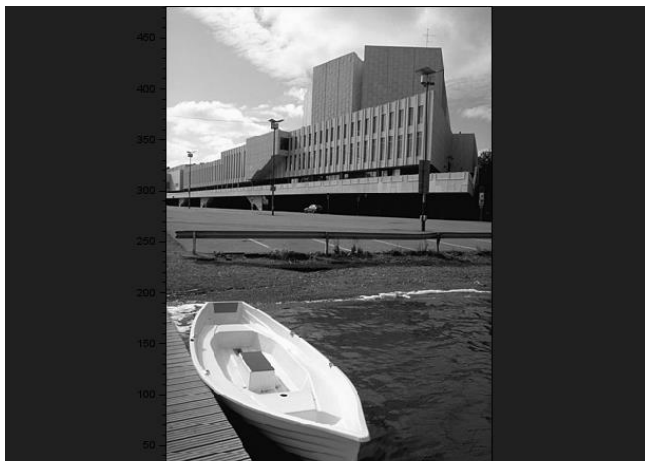
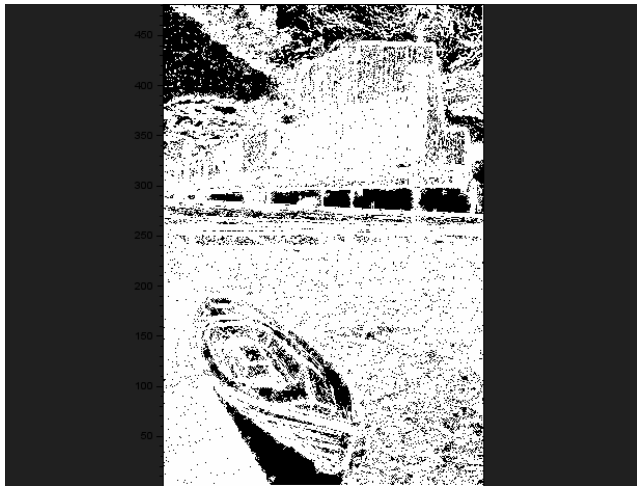
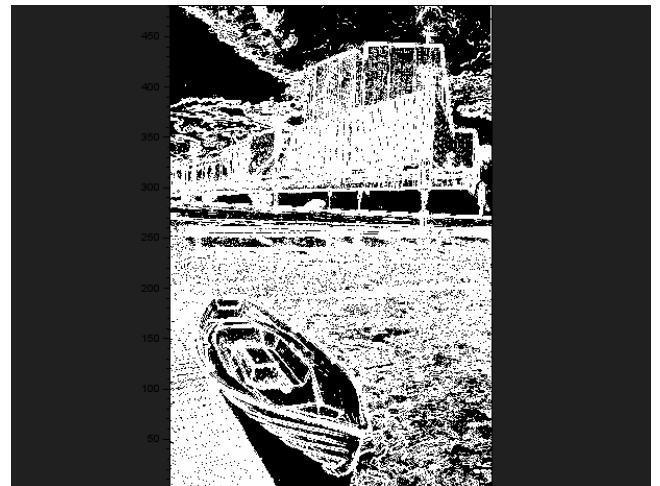


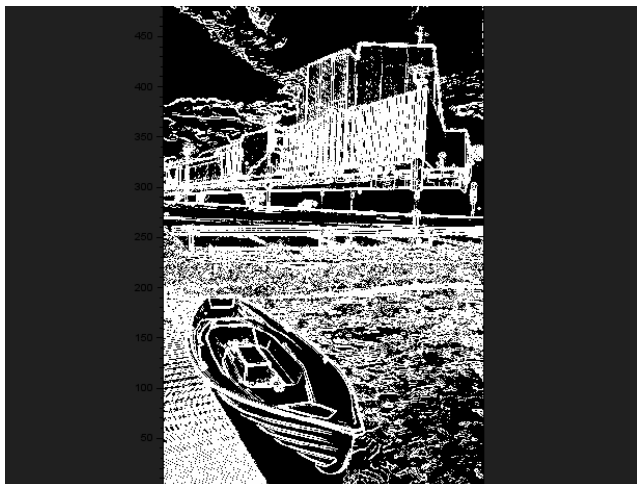
Image "barque.bmp"



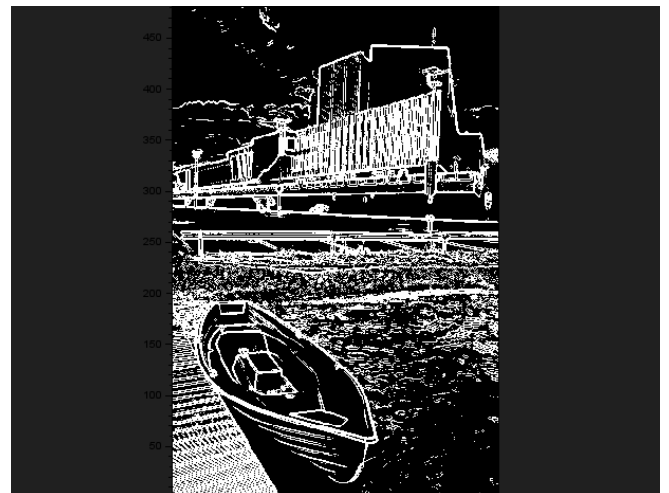
$P=0.1$  seuil=3



$P=0.3$  seuil=6



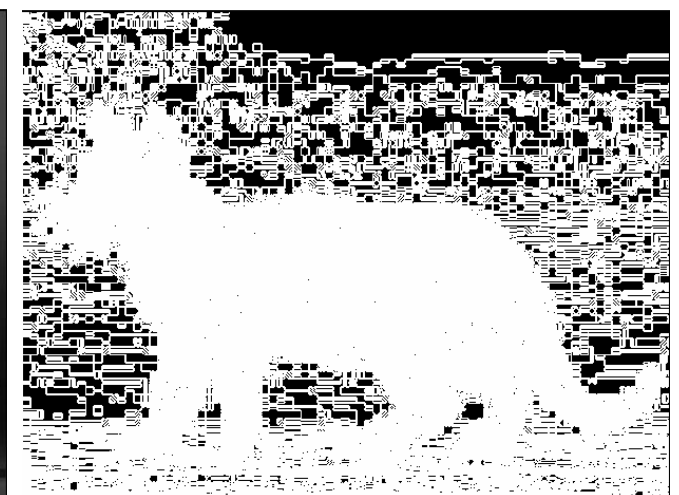
$P=0.5$  seuil=11



$P=0.7$  seuil=21



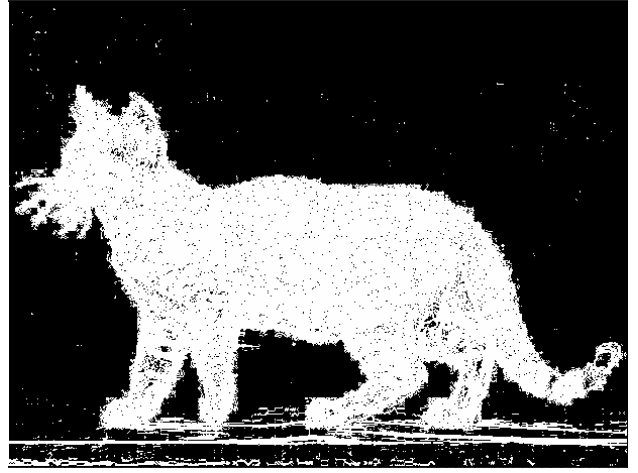
Image "chat.bmp"



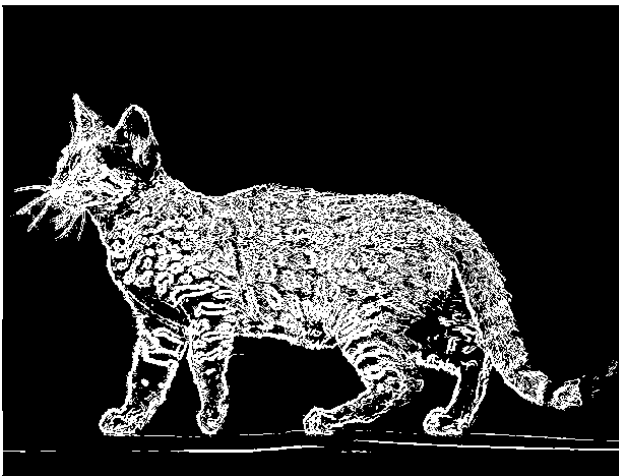
$P=0.2$  seuil=1



*P=0.4 seuil=3*



*P=0.6 seuil=5*


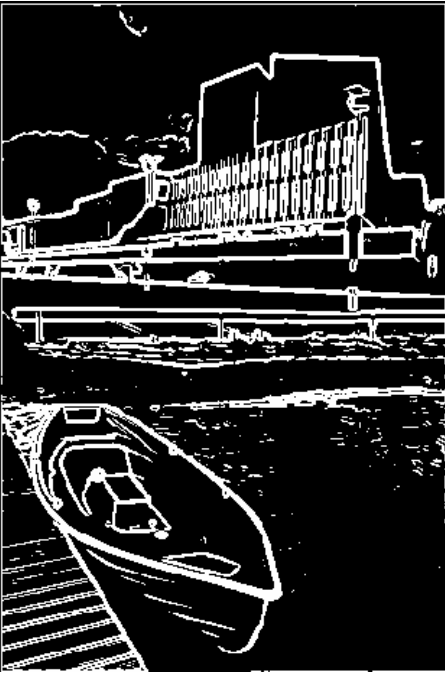



*P=0.8 seuil=23*

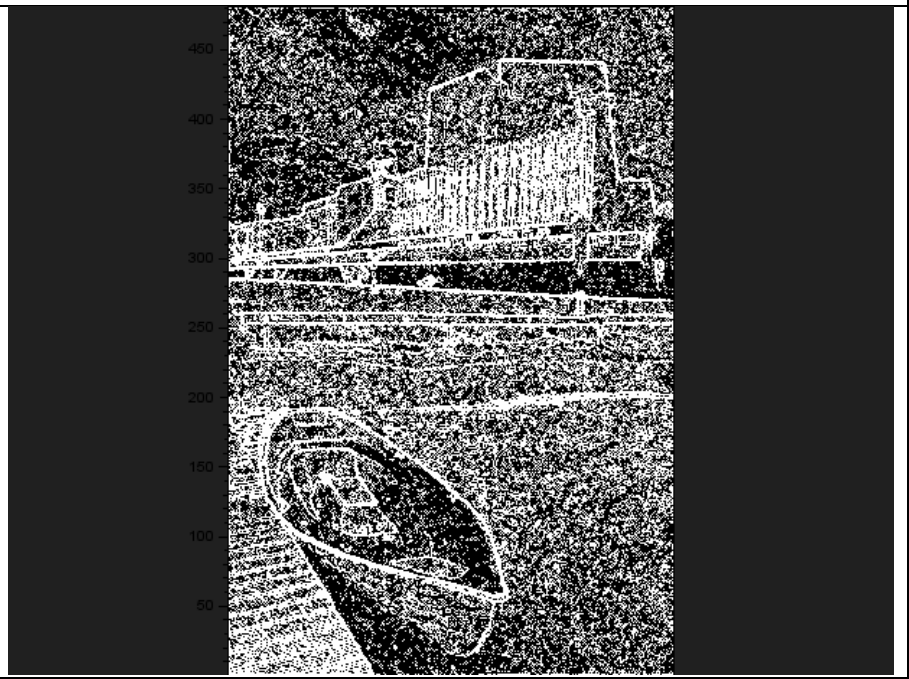
En observant les résultats, il apparaît que le pourcentage de pixels sélectionnés pour les contours devrait normalement être supérieur à 70% ( $P=0,7$ ). En effet, un pourcentage plus faible risque d'inclure des informations non importantes (bruits). Ainsi, pour obtenir les meilleurs résultats, il est recommandé de choisir une valeur de  $P$  située entre 0,7 et 0,9.

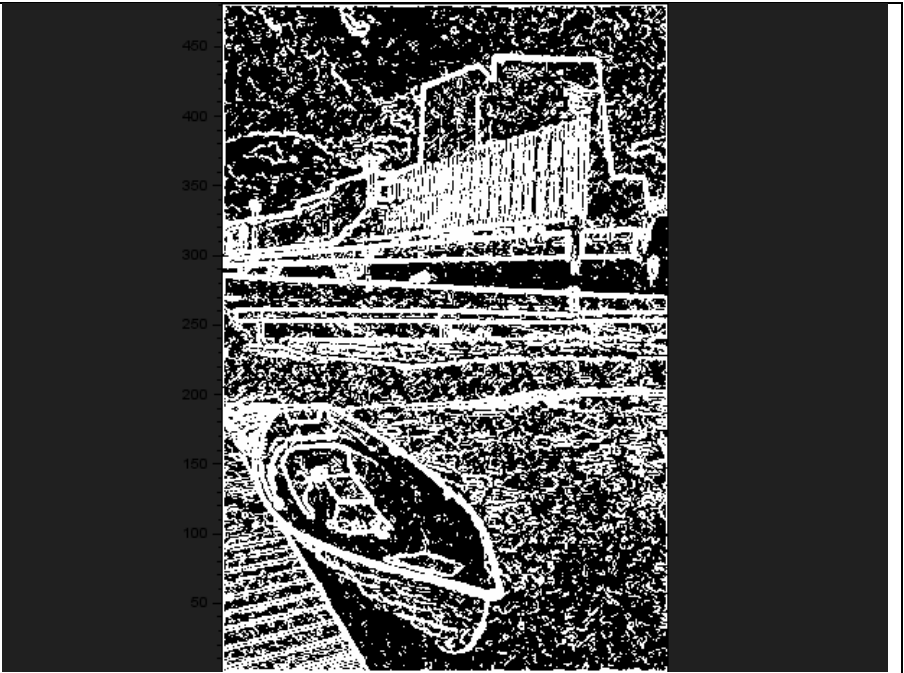
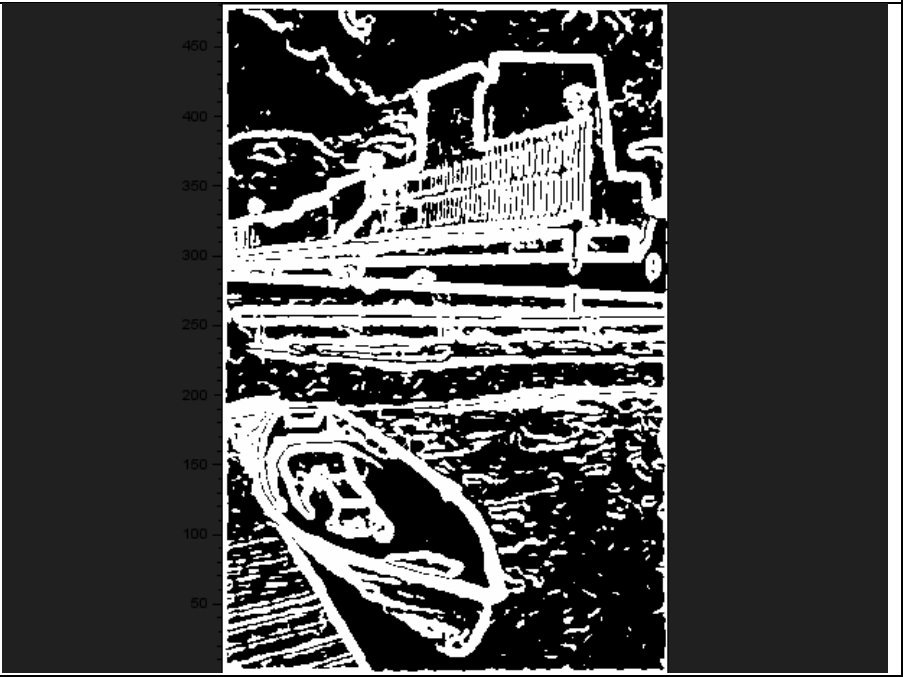
## #Exo4

	sigma	p	
--	-------	---	--



barque.bmp	0.5	0.8	 <p>Edge detection result for 'barque.bmp' using a threshold of 0.5. The image shows a boat in the foreground and a building in the background, with edges highlighted in white against a black background. A vertical axis on the left side of the image is labeled from 50 to 450 in increments of 50.</p>
	1	0.8	 <p>Edge detection result for 'barque.bmp' using a threshold of 1. The image shows the same boat and building, but with fewer edges detected compared to the 0.5 threshold. A vertical axis on the left side of the image is labeled from 50 to 450 in increments of 50.</p>


	2	0.8	
--	---	-----	--


	sigma	p	
barque_bruit.bmp	0.5	0.5	



	1	0.5	
	2	0.5	



	sigma	p	
plage.bmp	0.5	0.2	
	1	0.2	

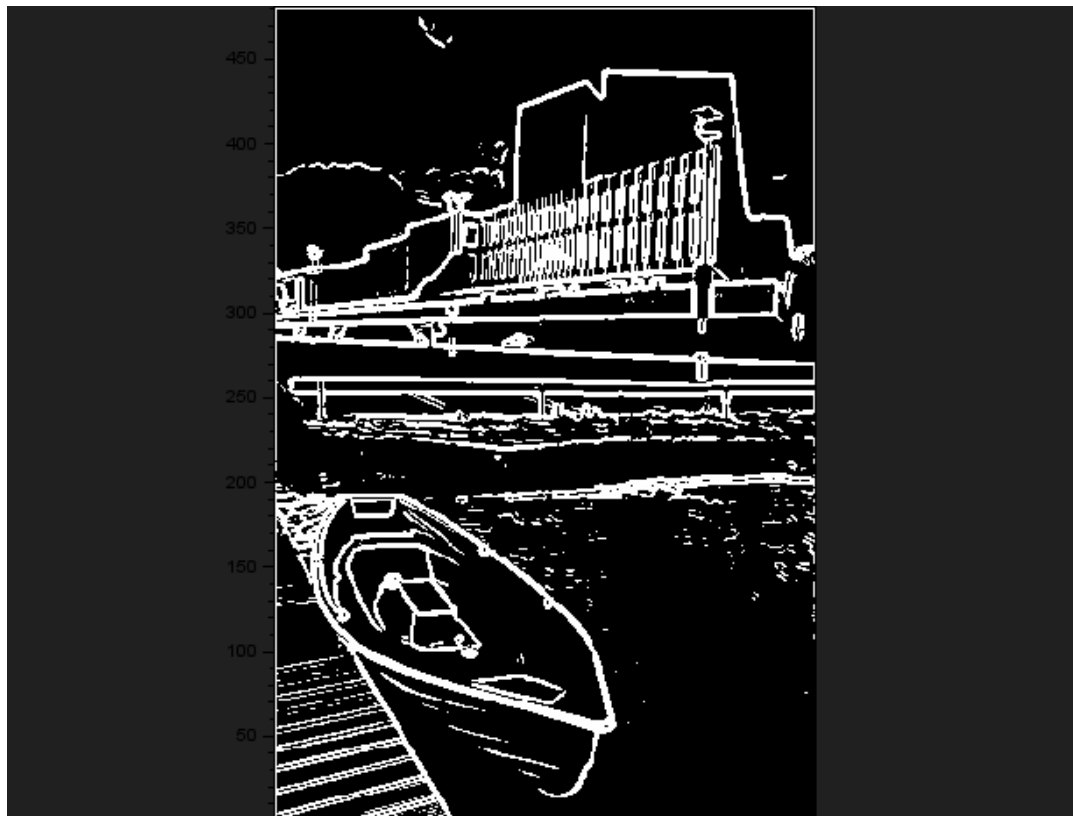
	2	0.2	
--	---	-----	--

	sigma	p	
sweets.bmp	0.5	0.7	

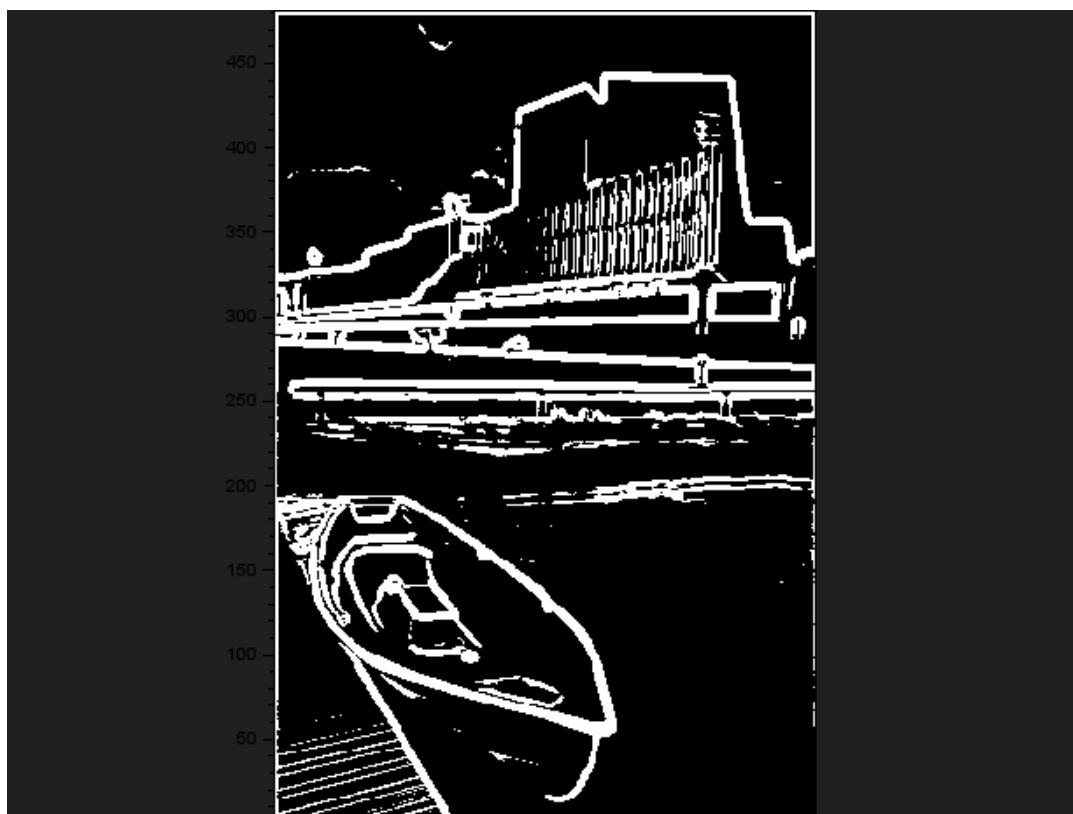
	1	0.7	
	2	0.7	

## #Exo5

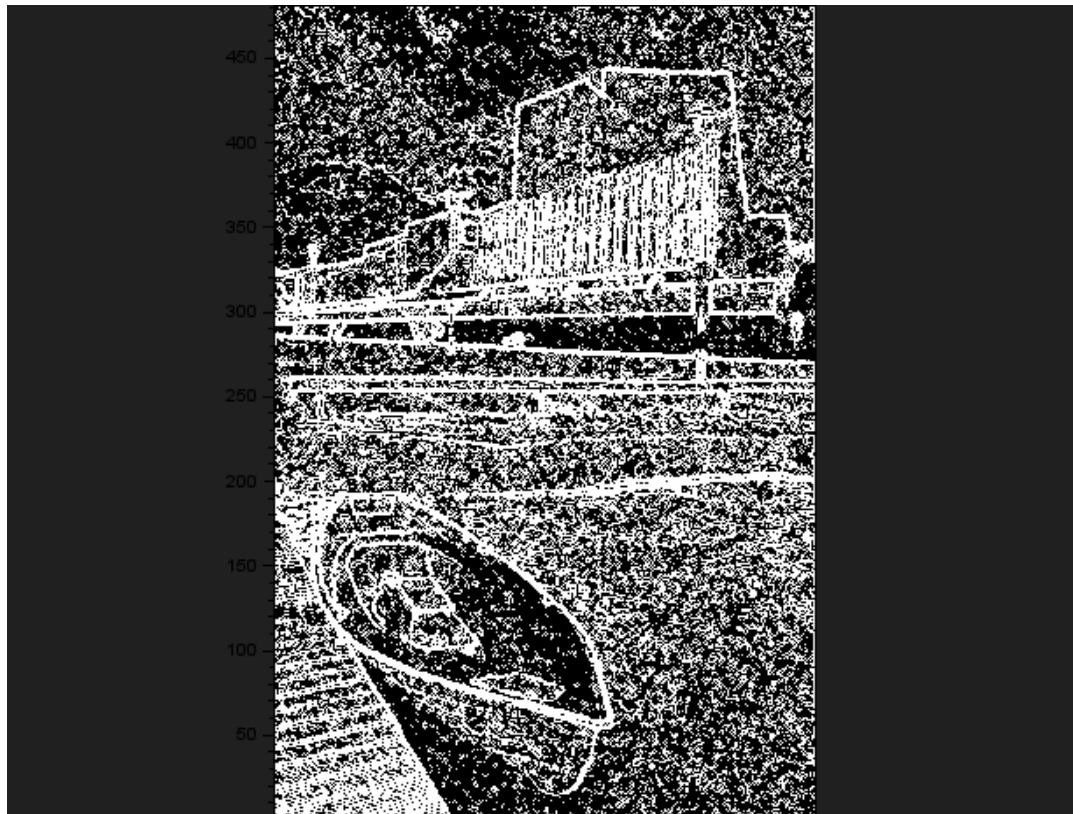
- Filtre de moyennage uniforme de taille 3x3 ( $\rho=0.8$ , barque.bmp)



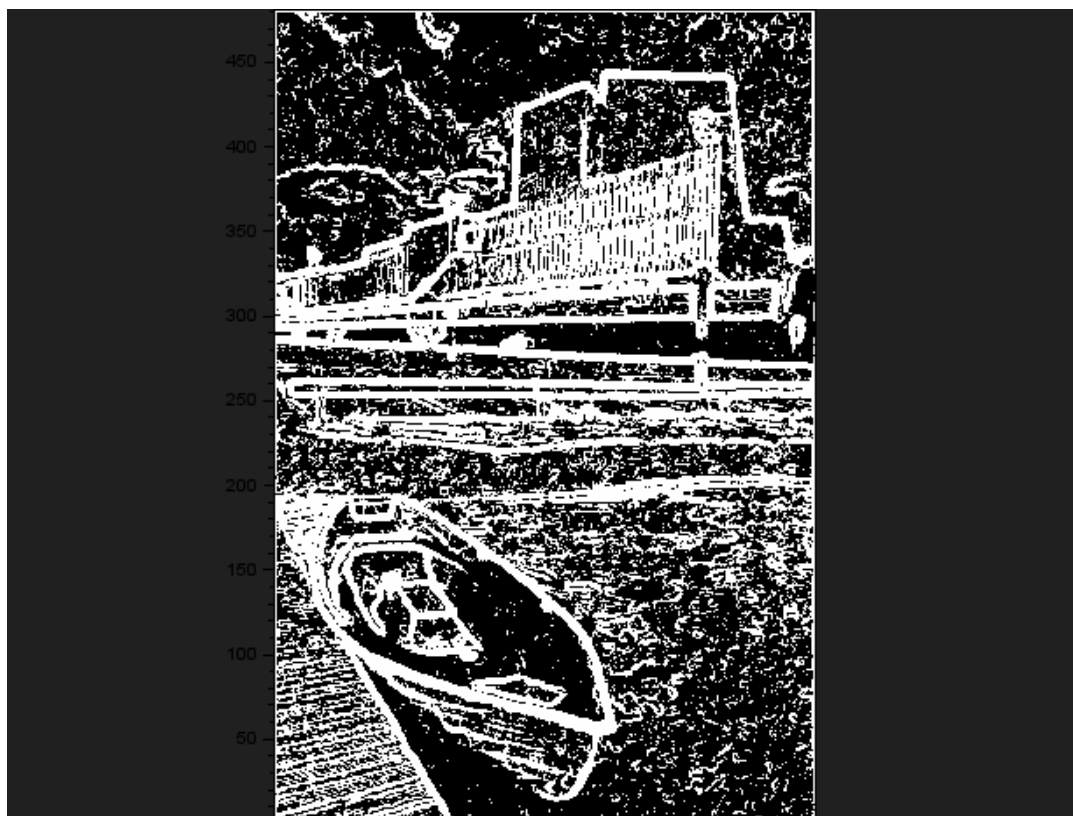
- Filtre de moyennage uniforme de taille 5x5 ( $\rho=0.8$ , barque.bmp)



- Filtre de moyennage uniforme de taille 2x2 ( $p=0.5$ , barque\_bruit.bmp)



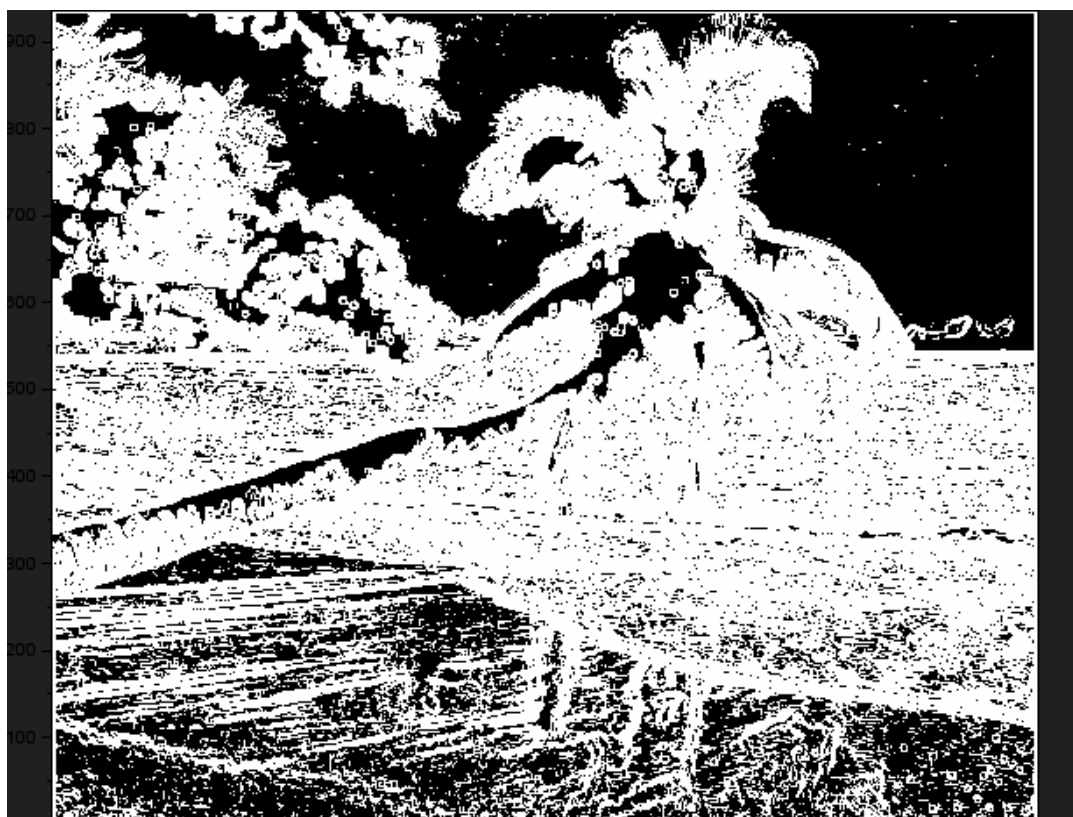
- Filtre de moyennage uniforme de taille 4x4 ( $p=0.5$ , barque\_bruit.bmp)



❖ Filtre de moyennage uniforme de taille 1x1 ( $p=0.2$ , plage.bmp)



❖ Filtre de moyennage uniforme de taille 7x7 ( $p=0.2$ , plage.bmp)



- Filtre de moyennage uniforme de taille 6x6 ( $p=0.7$ , sweets.bmp)



- Filtre de moyennage uniforme de taille 8x8 ( $p=0.7$ , sweets.bmp)



En modifiant la valeur de  $\sigma$ , on modifie la largeur de la fenêtre de lissage de la fonction gaussienne. Si  $\sigma$  est petit, la fenêtre sera étroite, et donc les pixels voisins auront moins d'influence sur la valeur du pixel en question, ce qui rendra le lissage plus local. En revanche, si  $\sigma$  est grand, la fenêtre sera large, et donc les pixels voisins auront plus d'influence sur la valeur du pixel en question, ce qui rendra le lissage plus global.

Si  $\sigma$  est trop petit, les contours peuvent être conservés de manière satisfaisante, mais le bruit sera plus présent. Si  $\sigma$  est trop grand, le lissage sera trop important et les contours importants de l'image risquent d'être perdus. Il est donc important de choisir une valeur appropriée de  $\sigma$  en fonction de la nature de l'image et de l'objectif de traitement.

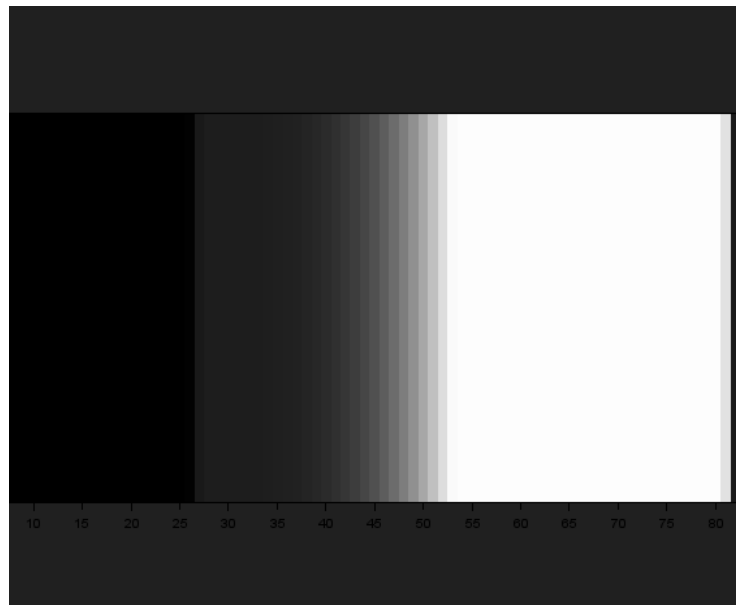
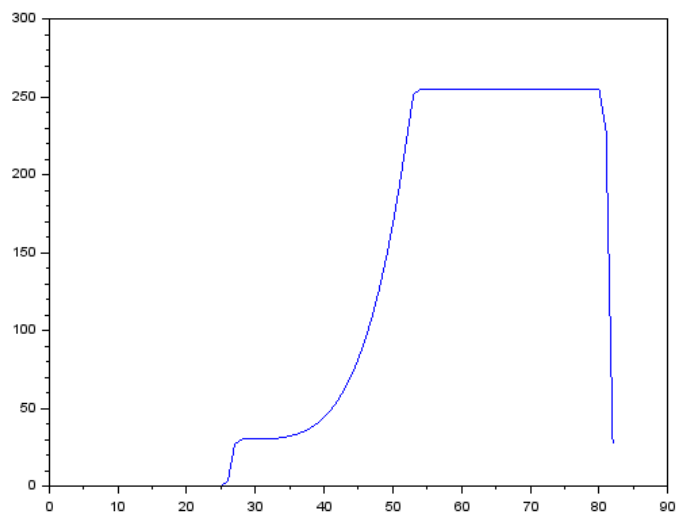
## #Exo5

En comparant les résultats de la détection de contours après filtrage Gaussien et après filtre de moyennage uniforme, on peut constater que le filtrage Gaussien donne de meilleurs résultats pour la détection de contours. En effet, les contours sont plus nets et mieux définis avec le filtre Gaussien, tandis que le filtre de moyennage uniforme a tendance à estomper les contours et à les rendre moins précis.

## #Exo6

1. En appliquant la fonction `contours_p` avec différentes valeurs de seuil  $p$ , on constate que pour un seuil trop élevé, certains contours ne sont pas détectés, tandis que pour un seuil trop bas, le bruit est également détecté, ce qui conduit à des contours imprécis. Il est difficile de trouver une valeur de seuil unique qui permette de détecter les deux contours souhaités avec une épaisseur comparable.
2. Lorsque l'on visualise la norme du gradient de l'image, on constate que les pics de la norme du gradient ne sont pas de la même hauteur en raison de la variation importante de l'intensité de l'image. Par conséquent, si l'on choisit un seuil unique pour la norme du gradient, on risque de manquer des contours importants. Une solution serait de normaliser l'image avant de détecter les contours afin de mettre en évidence tous les changements d'intensité importants avec la même échelle.





$P=0.1$



$P=0.2$



$P=0.3$



$P=0.4$

...



$P=0.82$



$P=0.9$



$P=0.95$

En effet, dans cette image, nous avons créé des variations de couleur au milieu de l'image (dans l'intervalle  $M/3$  et  $3M/3$ ). L'intensité des variations augmente en avançant horizontalement dans l'image, et la dernière variation est la plus grande. Ainsi, en regardant la norme du gradient de cette image dans cet intervalle sur une ligne (un signal), on peut observer que c'est une fonction croissante.

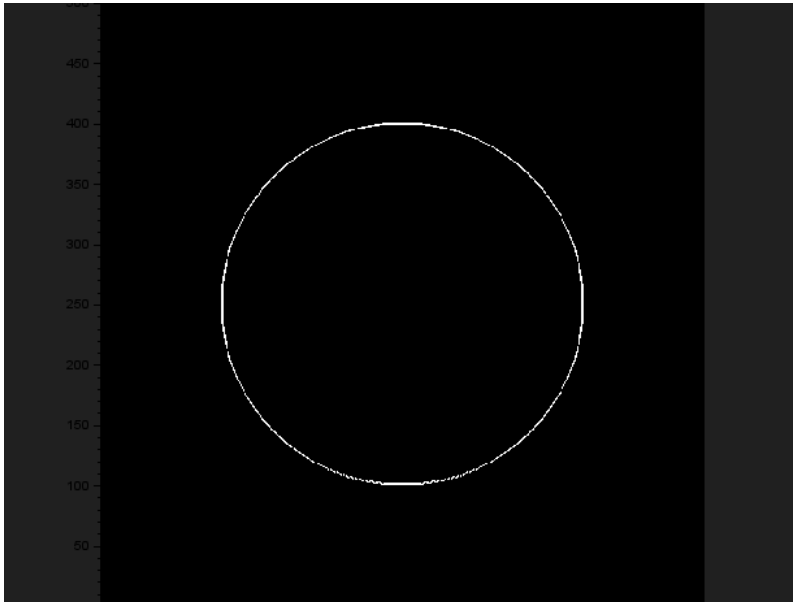
En augmentant le paramètre  $p$  pour trouver le seuil, on constate que le contour devient plus petit. En fixant  $p$  à 90, nous obtenons exactement ce que nous recherchions, c'est-à-dire l'emplacement où il y a la plus grande variation dans l'intervalle souhaité.

## #Exo8

```
exec('contours.sci', -1);
p = 0.8;
// im est l'image de l'exercice 1
afficher_image(contours_max(im, p));
```



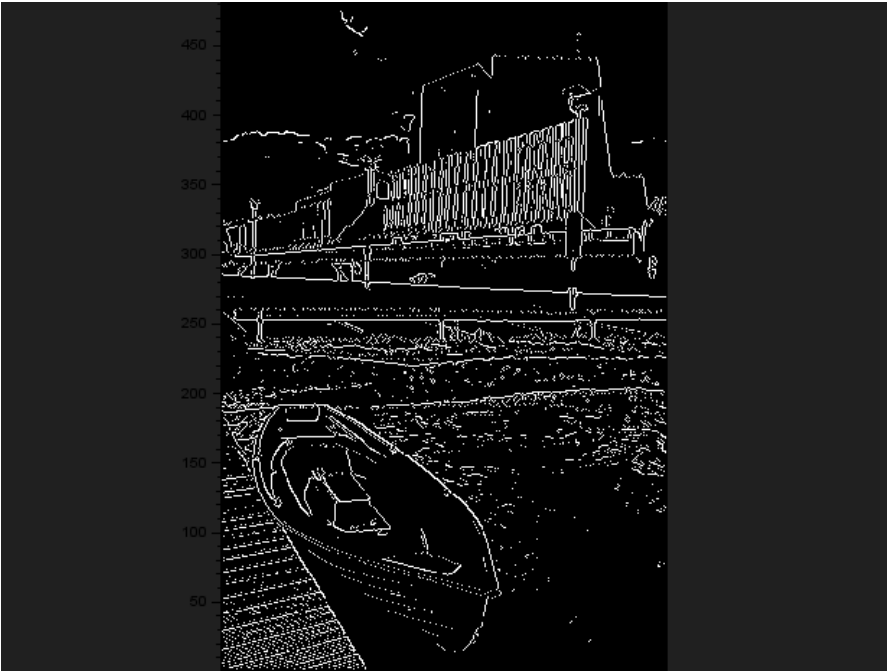
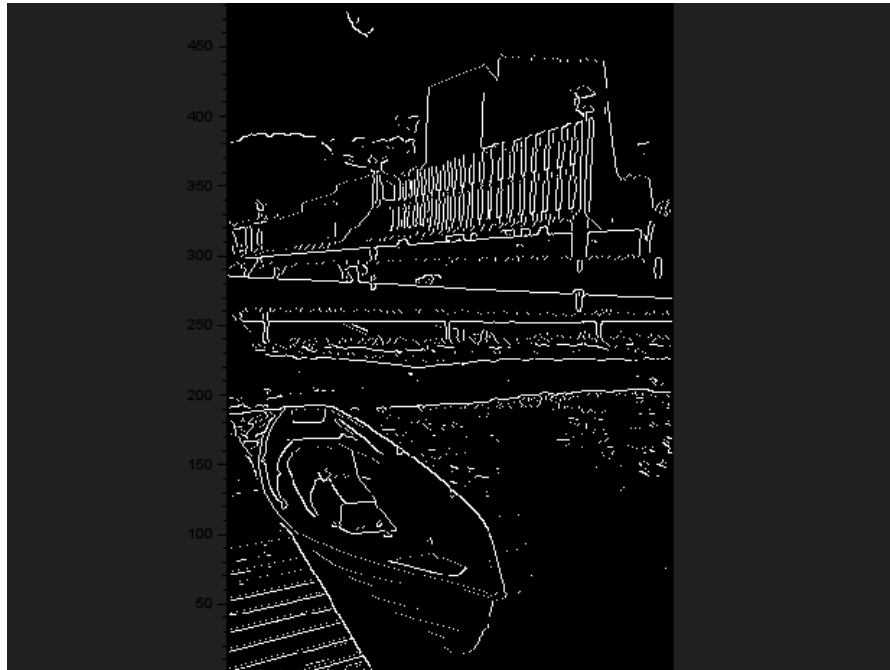
```
p = 0.8;  
// la fonction disque est fournie dans contours.sci  
im = disque();  
afficher_image(contours_max(im, p));
```

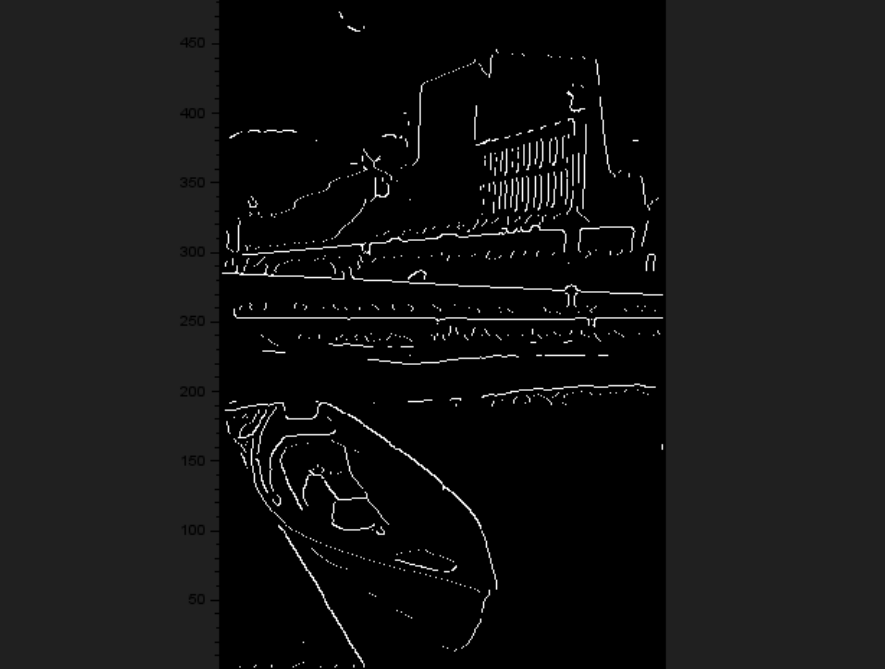


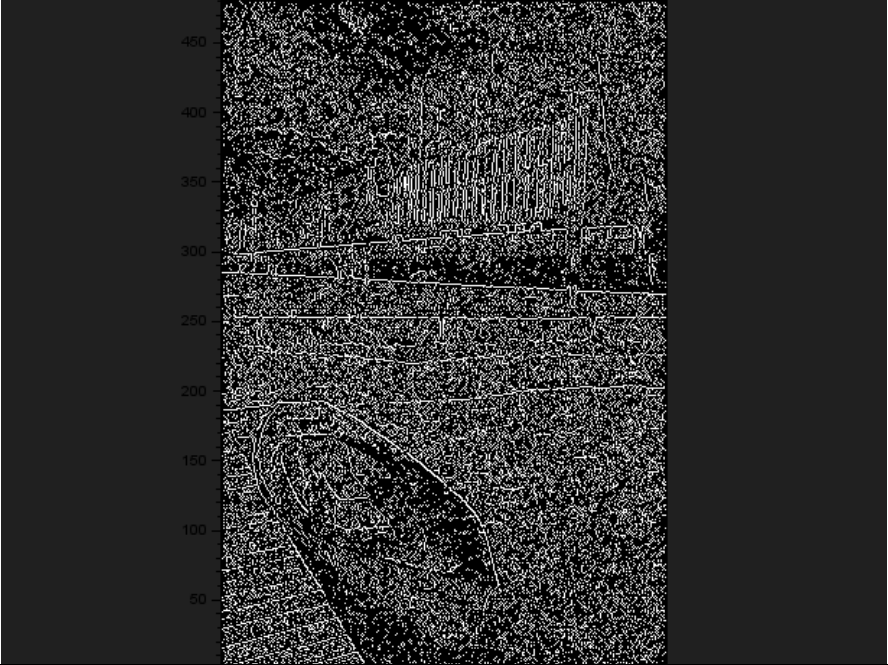
#Exo9

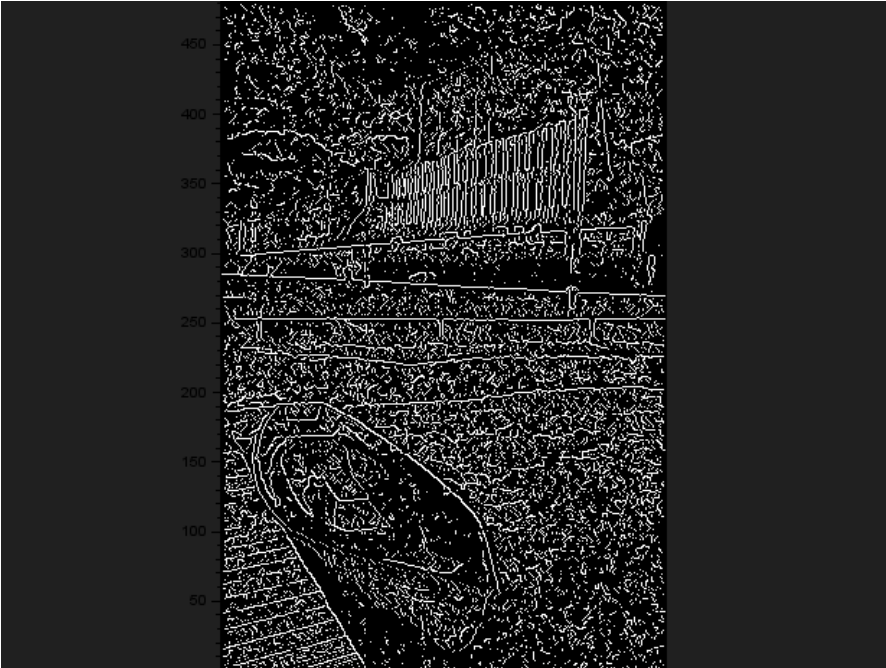
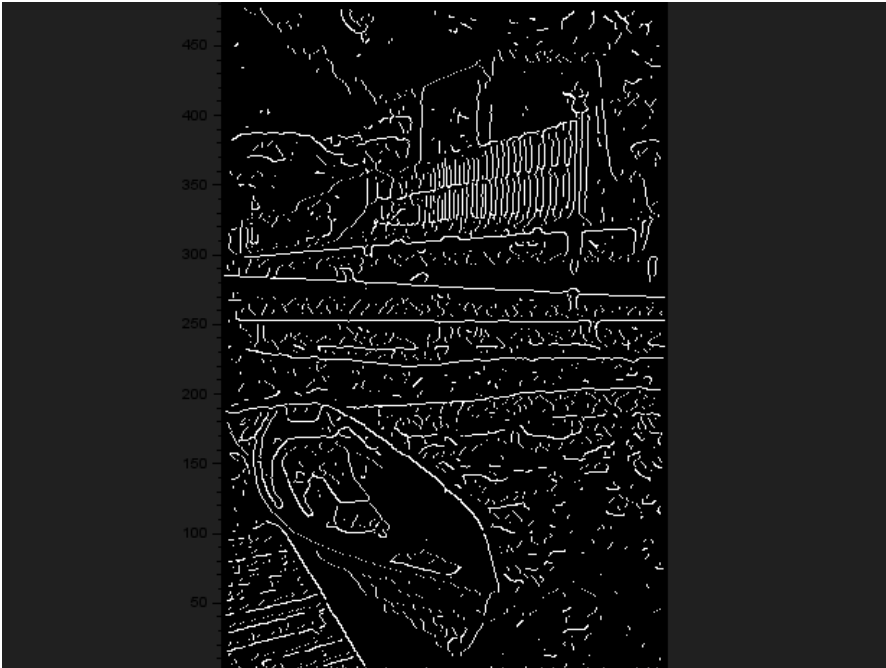
```
afficher_image(contours_max(conv2(im, W_gauss_2D(sigma), "same"), p));
```

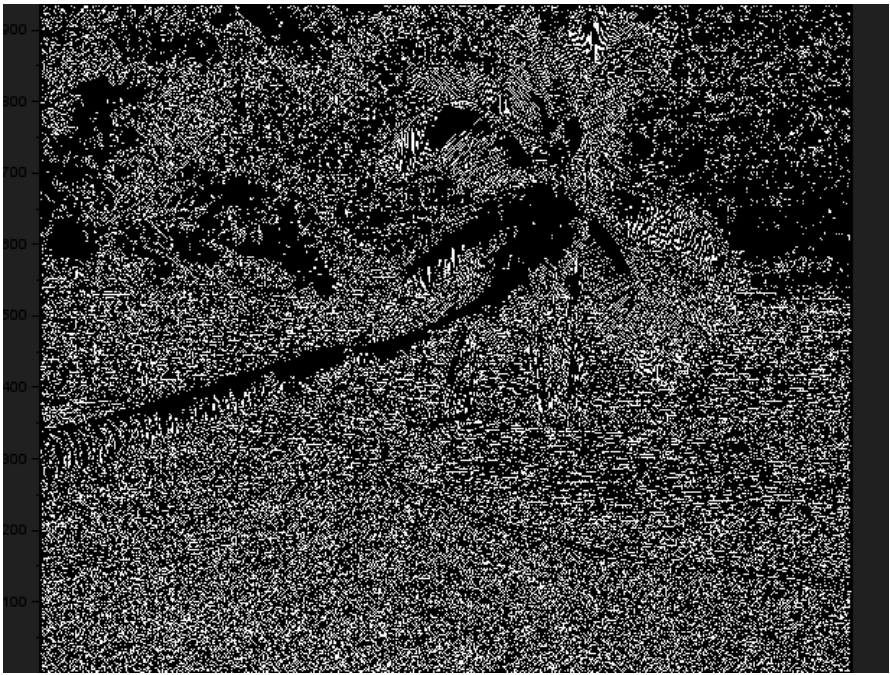

	sigma	p	
--	-------	---	--


barque.bmp	0.5	0.8	
	1	0.8	

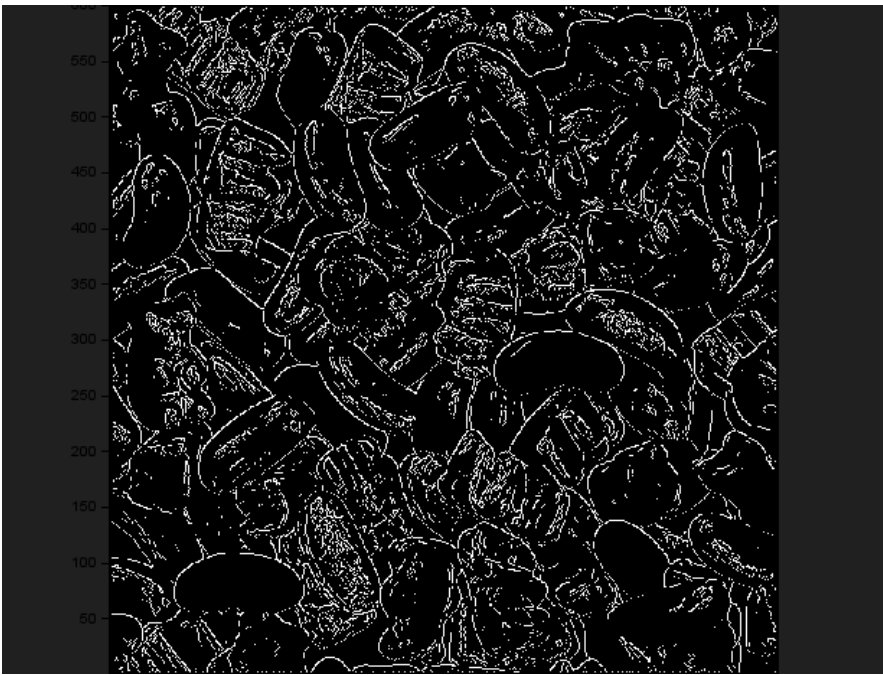
	2	0.8	
--	---	-----	--

	sigma	p	
barque_bruit.bmp	0.5	0.5	

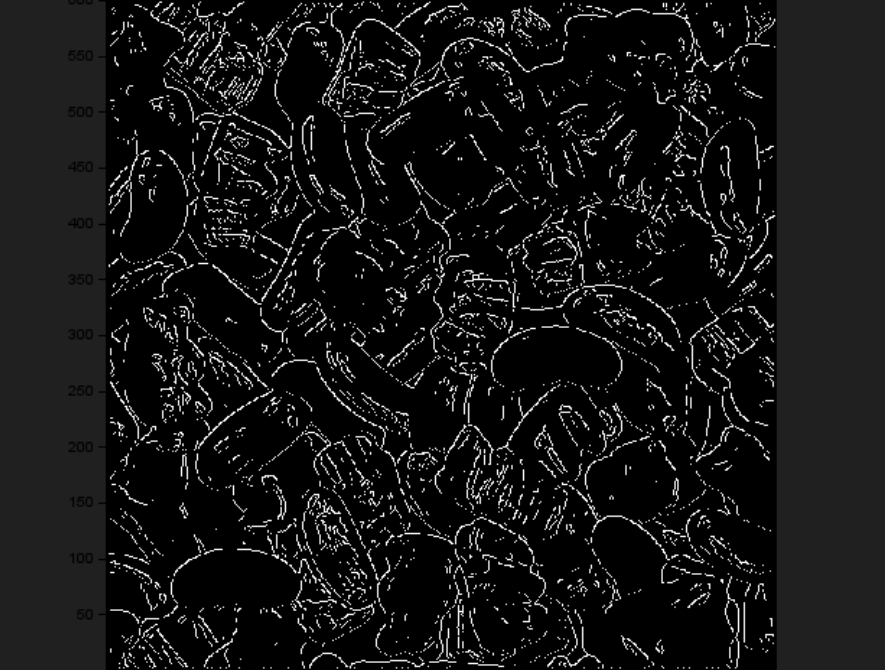
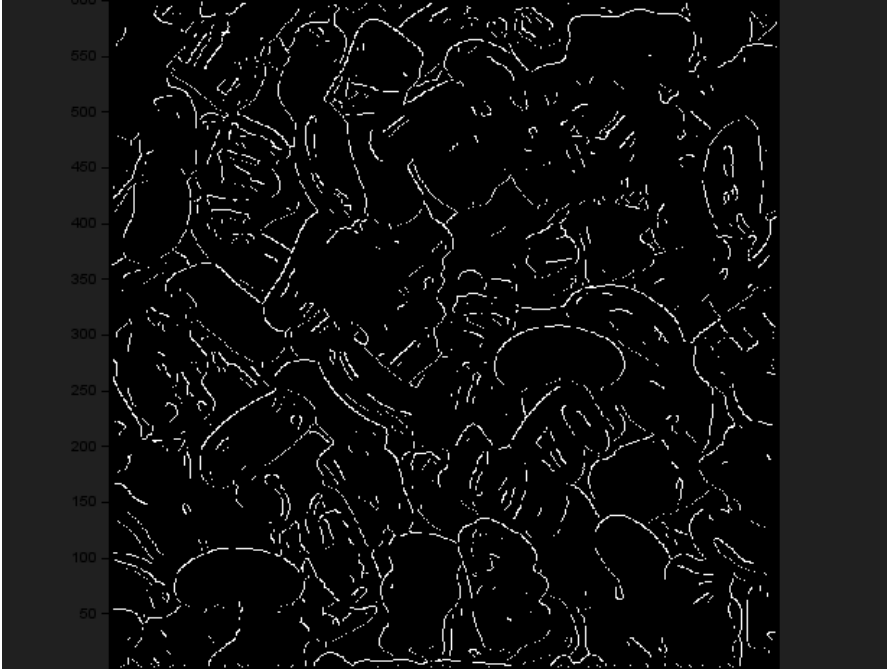
	1	0.5	
	2	0.5	

	sigma	p	
plage.bmp	0.5	0.2	
	1	0.2	

	2	0.2	
--	---	-----	--

	sigma	p	
sweets.bmp	0.5	0.7	



	1	0.7	
	2	0.7	

En effet, l'élimination des non maxima locaux permet de mieux mettre en évidence les contours en ne gardant que les points où le gradient est maximal dans la direction normale aux contours. Cela permet de ne pas avoir des contours trop épais et donc de mieux délimiter les objets présents dans l'image.

Cependant, cette méthode peut conduire à la suppression de certains contours fins ou mal alignés par rapport au gradient, ce qui peut entraîner des pertes d'informations. De plus, il est important de bien choisir le seuil de la magnitude du gradient pour ne pas supprimer des contours importants.

Contours.sci :

```
// calcule la norme du gradient d'une image
// le tableau imn en sortie peut comporter
// des valeurs non entieres
function imn=norme_gradient(im)
    // tableau imn initialise a la meme
    // taille que l'image
    imn = zeros(im);
    Dx = 0.5*[-1, 0, 1];
    Dy = 0.5*[1; 0; -1];
    imx = conv2(im, Dx, "same");
    imy = conv2(im, Dy, "same");
    imn=sqrt(imx.^2+imy.^2);
    imn = (imn - min(imn(:))) * 255 / (max(imn(:)) - min(imn(:)));
endfunction

/*exec('init_tp_image.sce');
exec('contours.sci');
im = lire_imageBMPgris('sweets.bmp');
imn = norme_gradient(im);
afficher_image(int(imn));*/

// determine les contours :
// valeur = 0 partout
// sauf pour pixels ou la norme du gradient est superieure
// a un seuil, dans ce cas valeur = 255
function imc=contours_seuil(im, seuil)
    imn = norme_gradient(im);
    imc = zeros(im);
    [M,N]=size(imc)
    for u=1:M
        for v=1:N
            if imn(u,v)< seuil then
                imc(u,v)=0
            else
                imc(u,v)=255
            end
        end
    end
endfunction
/*exec('contours.sci', -1);
im = lire_imageBMPgris('barque.bmp');
seuil = 20;
imc = contours_seuil(im, seuil);
afficher_image(imc);*/
```

```

// identifie le seuil tel qu'il y a
// un pourcentage p des pixels pour
// lesquels la norme du gradient est
// inferieure a ce seuil
function seuil=trouver_seuil(im, p)
    G = norme_gradient(im);
    H = hist_cumul(G);
    seuil = 1;
    while H(seuil)/H(256) < p
        seuil = seuil + 1;
    end
endfunction

/*exec('init_tp_image.sce');
exec('contours.sci', -1);
p = 0.8;
im = lire_imageBMPgris('barque.bmp');
seuil = trouver_seuil(im, p);
disp(seuil);*/

//pour p=0,8 seuil=31
//pour p=0,5 seuil=11
//pour p=0 seuil=256
//pour p=1 seuil=1
/*
im = lire_imageBMPgris('barque.bmp');
p = 0.9;
afficher_image(contours_p(im, p));
seuil = trouver_seuil(im, p);
disp(seuil);
*/
// determine le seuil a partir d'un
// pourcentage p
function imc=contours_p(im, p)
    G = norme_gradient(im);
    imc = contours_seuil(im, trouver_seuil(im, p));
endfunction

// filtre Gaussien
/*exec('filtres2D.sci');
p = 0.8;
im = lire_imageBMPgris('barque.bmp');
sigma = 0.5;*/
//afficher_image(contours_p(conv2(im, W_gauss_2D(sigma), "same"), p));

// filtre moyenne
/*M = 1/9*ones(8, 8);*/
//afficher_image(contours_p(conv2(im, M, "same"), p));

exec('filtres1D.sci');

// image initialement noire
N = 40;

```

```

M = 2*N
im = zeros(N,M);

// l'image est compos'ee de trois bandes de taille n:
n = floor(M/3)

// le premier tiers reste noir
// le deuxi`eme tiers est un d'egrad'e avec un fort gradient
// de 30 `a 255
for j = n:2*n
    im(:,j) = 30 + ((j - n)/n)^4*225;
end

// le dernier tiers est `a 255
im(:, 2*n+1:M) = 255;

// lissage gaussien
G = W_gauss(0.5);
im = conv2(im, G);

// extraction des contours
contours = contours_p(im, 0.95);

// affichage des contours
//afficher_image(contours);

// tracé de la ligne médiane de l'image pour visualiser l'épaisseur des contours
//scf();
//plot(contours(N/2, :));

// calcul de l'histogramme cumule
function Hist=hist_cumul(im)
    // calcul de l'histogramme
    classes = [-0.1, linspace(0,255,256)];
    hist = histc(classes, im, normalization=%f);
    Hist = zeros(256)
    Hist(1) = hist(1)
    for i = 2:256
        Hist(i) = Hist(i-1) + hist(i)
    end
endfunction

//afficher_image(contours_p(conv2(im, W_gauss_2D(sigma),"same"),p));

function [u1, v1, u2, v2]=indices_voisins(u, v, phi)
    //ici en fonction de l'angle donné on définit la direction du gradient et on retourne les indice
    //des voisins avant et après dans cette direction
    if (phi <= %pi/8) || (phi >= 7*%pi/8) then //cas 1
        u1=u ; u2=u ; v1=v-1 ; v2=v+1
    elseif (phi > %pi/8) && (phi <= 3*%pi/8) then //cas2
        u1=u+1 ; u2=u-1 ; v1=v-1 ; v2=v+1
    elseif (phi > 3*%pi/8) && (phi < 5*%pi/8) then //cas3
        u1=u-1 ; u2=u+1 ; v1=v ; v2=v
    else //cas 4

```

```

    u1=u+1 ; u2=u-1 ; v1=v+1 ; v2=v-1
end
endfunction
// renvoie une image 500x500 avec un disque blanc
// sur fond noir
function im_out=contours_max(im, p)
/* dans cette fonction ,pour chaque pixel
on trouve d'abord l'angle de dérivé horizontalement et de dérivé verticalement
et puis en utilisant la fonction précédent on récupère les coordonnées des voisins
dans la direction de gradient et on verifie si tous les deux voisins ont une norme gradient
inférieur à la norme de gradient du pixel central, on sait qu'on est dans le pique et on met à
255 la valeur de cette pixels et 0 les autres
*/

[M,N]=size(im)
im_out=zeros(M,N)
apres_p=contours_p(im,p)
imn = norme_gradient(im)
Dx=[-1,0,1]
Dy=[1,0,-1]
imx=conv2(im,Dx,"same")
imy=conv2(im,Dy,"same")
for i=2:M-1
    for j=2:N-1
        if apres_p(i,j)==255 then
            phi=atan(imy(i,j),imx(i,j))
            [u1,v1,u2,v2] = indices_voisins(i,j,phi)
            if imn(u1,v1)<= imn(i,j) && imn(u2,v2)<= imn(i,j) then
                im_out(i,j)=255
            else im_out(i,j)=0
            end
        else im_out(i,j)=0
        end
    end
end
endfunction

function im=disque()
im = zeros(500, 500)
for i = 1:500
    for j = 1:500
        if abs(i-250)^2 + abs(j-250)^2 < 150^2 then
            im(i,j) = 255
        end
    end
end
endfunction
exec('filtres2D.sci');
p = 0.7;
im = lire_imageBMPgrise('sweets.bmp');
sigma = 2;
afficher_image(contours_max(conv2(im, W_gauss_2D(sigma),"same"),p));

```