Détection des bords

1. Introduction

La détection des bords dans une image est souvent la première étape d'un algorithme d'extraction d'informations, par exemple la détection de formes.

Ce document traite la détection de bords par dérivation du premier ordre. Un bord est une variation importante du niveau de gris localisée sur une petite région de l'image. Il est associé à une relativement grande dérivée. La première étape est donc le calcul du gradient par différentiation. Lorsque l'image est bruitée, il est nécessaire d'associer la différentiation à un lissage gaussien (filtre passe-bas). Nous allons détailler la méthode de Canny ([1]), constituée des étapes suivantes :

- ▷ lissage Gaussien ;
- ▷ opérateur de différentiation de Sobel ;
- ▷ suppression des non-maxima ;
- ▷ seuillage ;

2. Calcul du gradient

Si l'image est considérée comme un signal continu $v(x, y)$, il s'agit de calculer le gradient

$$\overrightarrow{grad}\, v = \frac{\partial v}{\partial x}\overrightarrow{e_x} + \frac{\partial v}{\partial y}\overrightarrow{e_y} \tag{1}$$

Les bords sont les points où la norme du gradient est relativement grande. De plus, le gradient est perpendiculaire au bord.

Pour calculer une approximation du gradient sur une image discrétisée, on doit appliquer un opérateur de différentiation. Soit $V_{i,j}$ la valeur du pixel (i, j). Une approximation de la dérivée par rapport à x en ce point peut être obtenue par une différence centrée :

$$V_{i+1,j} - V_{i-1,j} \tag{2}$$

En terme de filtrage, cela correspond au masque de convolution (ou réponse impulsionnelle) :

$$\begin{pmatrix} -1 & 0 & 1 \end{pmatrix} \tag{3}$$

Il est préférable d'appliquer un masque carré 3x3. Pour cela, la solution optimale est celle de l'opérateur de Sobel :

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \tag{4}$$

Ce masque effectue une différentation sur l'axe x associée à un lissage sur l'axe y. On définit de même l'opérateur de Sobel pour l'axe y :

$$S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{5}$$

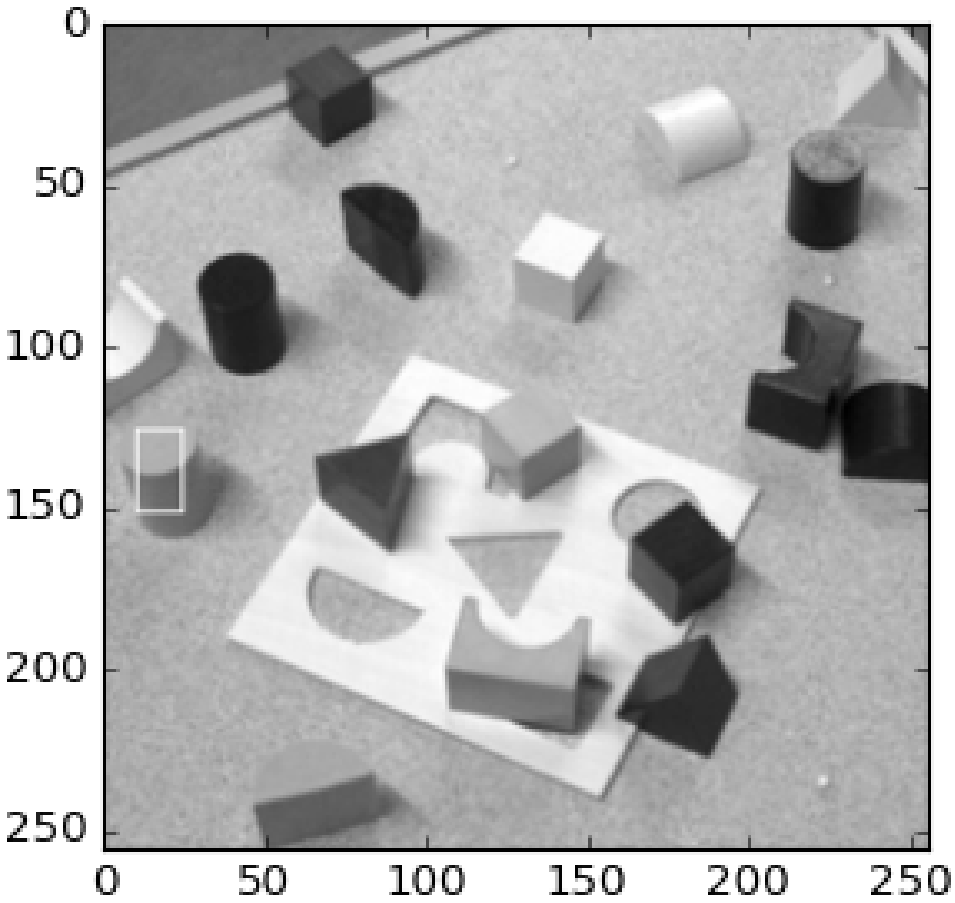
La seconde étape consiste à calculer la norme du gradient et l'angle par rapport à la direction x :

$$G = ||\overrightarrow{grad}\, v|| \tag{6}$$

$$\theta = (\overrightarrow{e_x}, \overrightarrow{grad}\, v) \tag{7}$$

Voyons un exemple. La traitement est appliqué à la couche rouge d'une image RGBA. La fonction `matplotlib.pyplot.imread` fournit un tableaux de flottants compris entre 0 et 1.

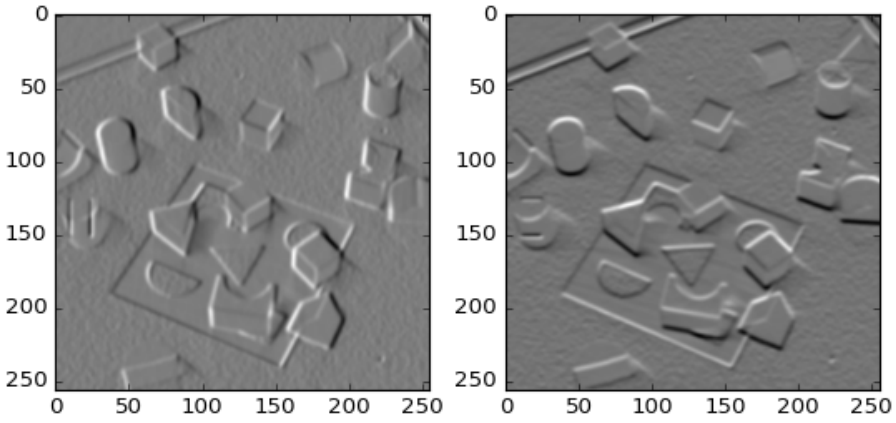
```
import numpy
import math
import cmath

from scipy.ndimage.filters import convolve,gaussian_filter
from  matplotlib.pyplot import *
img = imread("../../../../simul/image/objets.png")
red = img[:,:,0]
green = img[:,:,1]
blue = img[:,:,2]
array=red*1.0
figure(figsize=(4,4))
imshow(array,cmap=cm.gray)
```




```
array = gaussian_filter(array,1)
sobelX = numpy.array([[-1,0,1],[-2,0,2],[-1,0,1]])
sobelY = numpy.array([[-1,-2,-1],[0,0,0],[1,2,1]])
derivX = convolve(array,sobelX)
derivY = convolve(array,sobelY)
gradient = derivX+derivY*1j
G = numpy.absolute(gradient)
theta = numpy.angle(gradient)
```

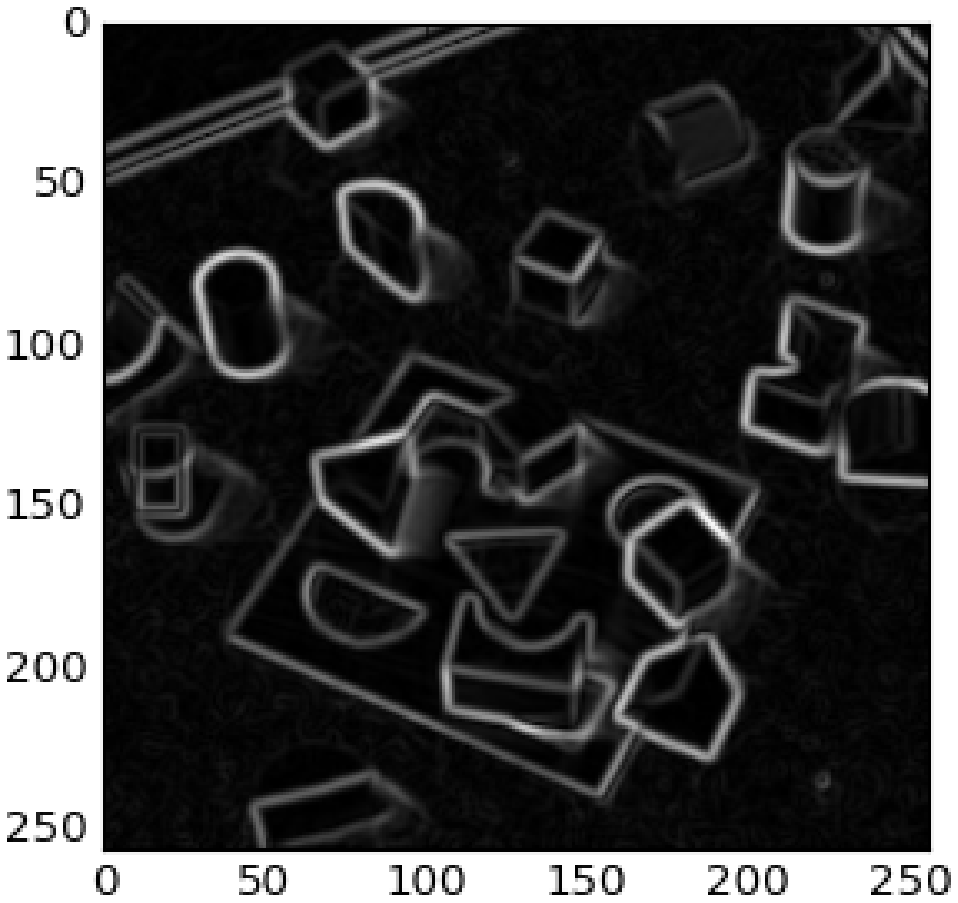
Représentation des composantes du gradient :

```
figure(figsize=(8,4))
f,(p1,p2)=subplots(ncols=2)
p1.imshow(derivX,cmap=cm.gray)
p2.imshow(derivY,cmap=cm.gray)
```



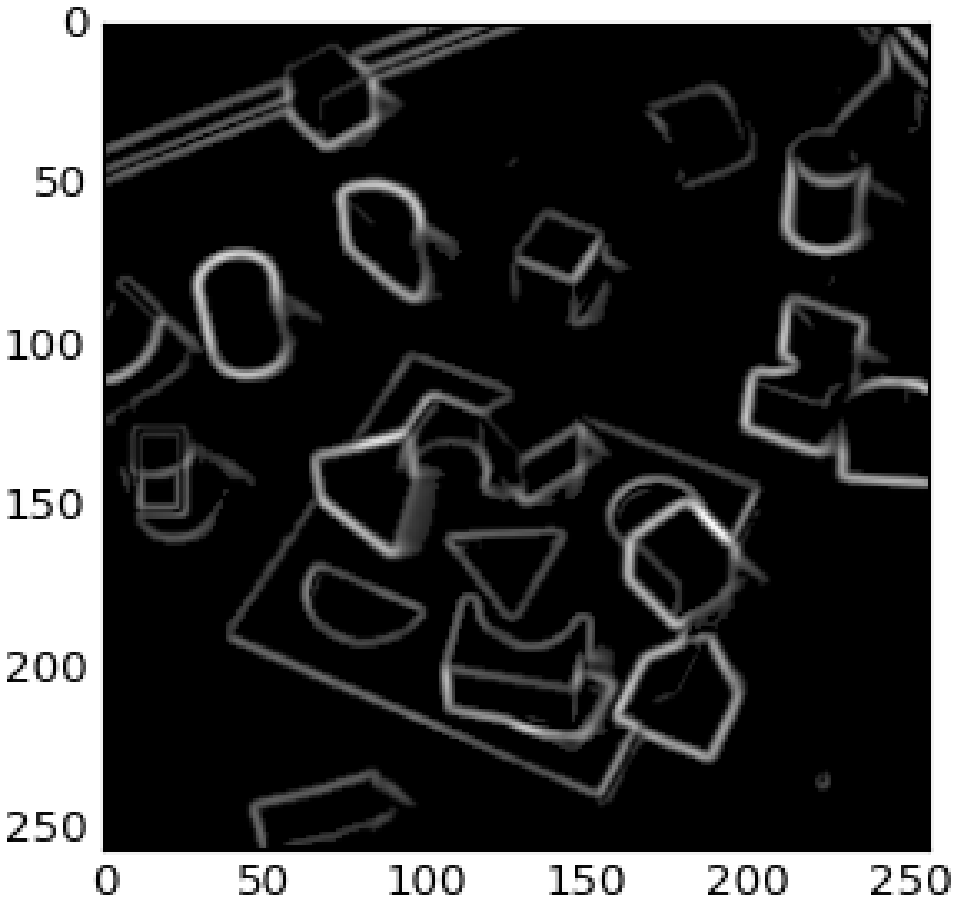
Représentation de la norme du gradient :

```
figure(figsize=(4,4))
imshow(G,cmap=cm.gray)
```



L'image de départ est relativement peu bruitée. On observe tout de même quelques maxima du gradient liés à la présence du bruit. L'étape suivante consiste à éliminer les points dont la valeur de G est en dessous d'un seuil :

```
seuil = 0.23
s = G.shape
for i in range(s[0]):
    for j in range(s[1]):
        if G[i][j]<seuil:
            G[i][j] = 0.0
figure(figsize=(4,4))
imshow(G,cmap=cm.gray)
```



La valeur de seuil à appliquer dépend du niveau de bruit dans l'image et de la netteté des bords que l'on souhaite détecter. Il s'agit donc d'un paramètre à régler au cas par cas.

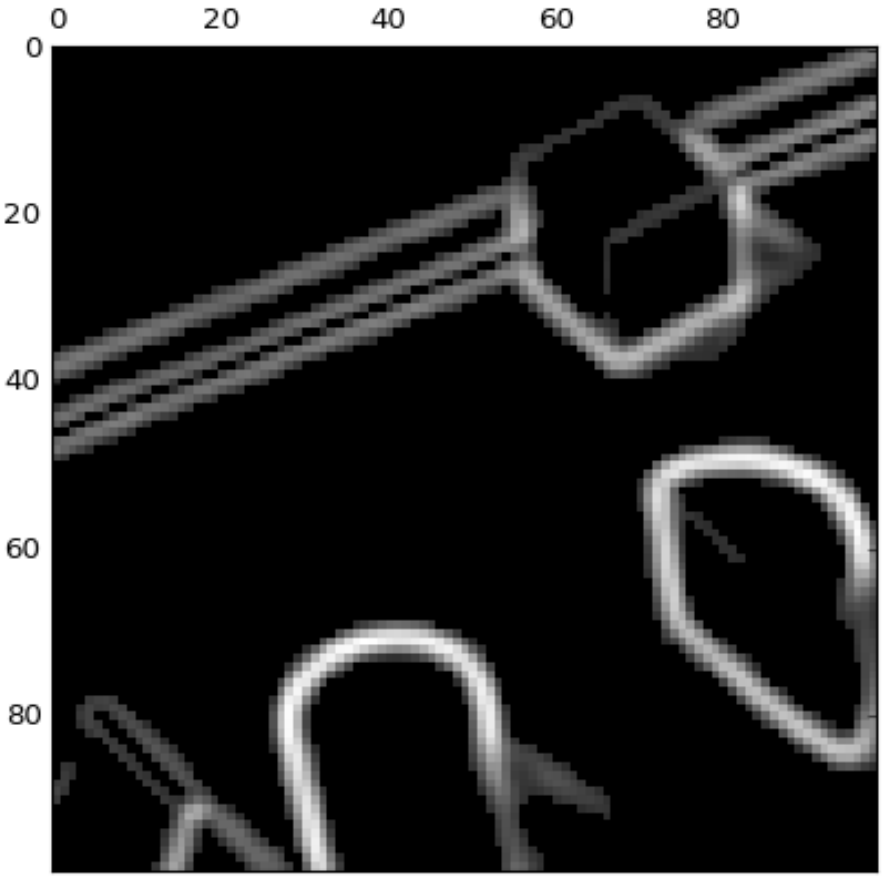
Dans le cas d'une image très bruitée, il peut être nécessaire d'appliquer au préalable un lissage gaussien.

On remarque que la ligne blanche rectangulaire présente dans l'image de départ se traduit pas la présence de deux lignes (la ligne a deux bords). Cela montre que la différentiation du premier ordre n'est pas adaptée à la détection des lignes.

3. Suppression des non-maxima

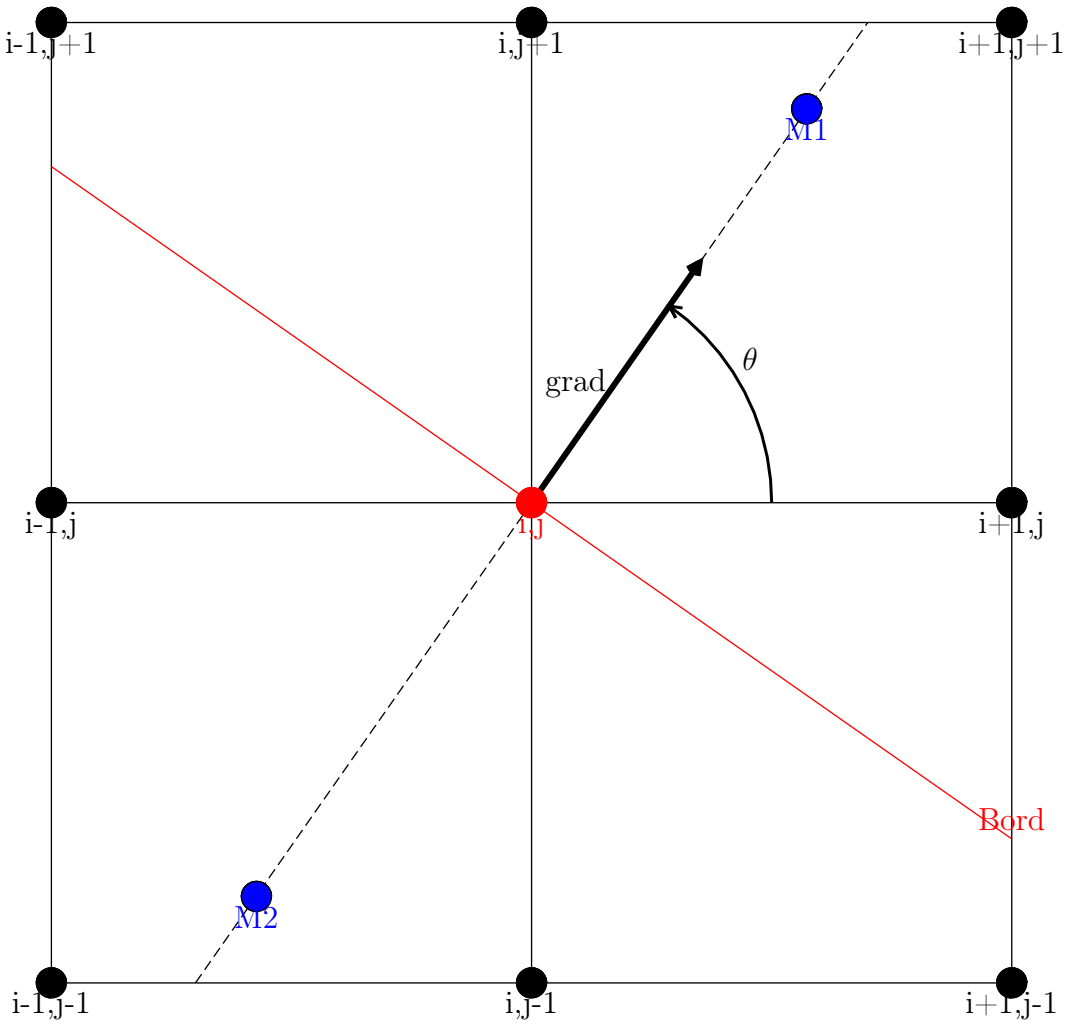
Observons un détail de l'image :

```
rect = G[1:100,1:100]
figure(figsize=(4,4))
matshow(rect,cmap=cm.gray)
```



Les bords obtenus ont une largeur de plusieurs pixels. Ici, la largeur est d'environ 3 pixels, mais elle peut être beaucoup plus grande si l'image de départ a subi un lissage important, ou si les bords sont moins nets.

L'étape suivante consiste à éliminer les pixels sur lesquels le gradient n'est pas maximal. La figure suivante représente un pixel (i, j) et ses 8 proches voisins. Le bord en ce point (trait plein) est perpendiculaire au gradient. On a aussi représenté deux points M_1 et M_2 situés sur la normale (trait pointillé), de part et d'autre du bord, à une distance unité du point (i, j).



La valeur de la norme du gradient G aux points M_1 et M_2 est calculée par interpolation bilinéaire à partir des 4 pixels voisins. Le pixel (i, j) est retenu si son gradient est supérieur à ceux des points M_1 et M_2. Dans le cas contraire, le pixel est éliminé. On élimine donc ainsi les pixels qui ne sont pas sur un maximum du gradient (maximum le long de la normale).

On commence par faire une copie de la matrice du gradient, dans laquelle les non-maxima seront enlevés :

```
Gmax = G.copy()
```

On utilise la fonction d'interpolation suivante :

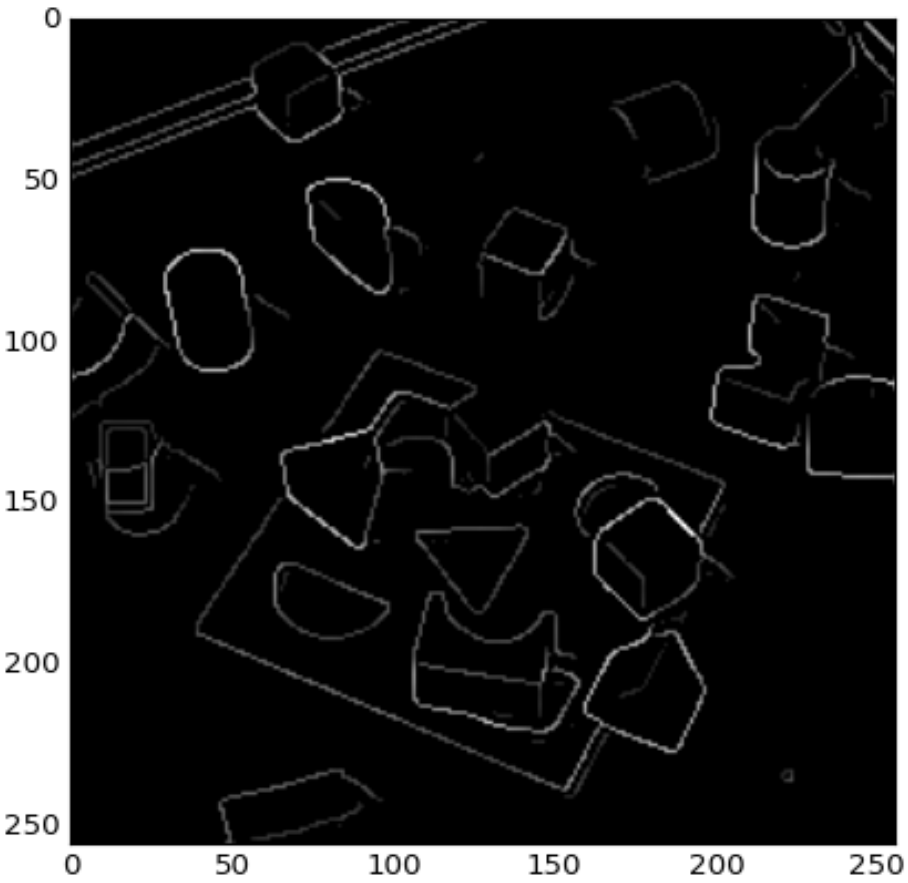
```
def interpolation(array,x,y):
    s = array.shape
    i = math.floor(x)
```

```
    j = math.floor(y)
    t = x-i
    u = y-j
    u1 = 1.0-u
    t1 = 1.0-t
    if j==s[0]-1:
        if i==s[1]-1:
            return array[j][i]
        return t*array[j][i]+t1*array[j+1][i]
    if i==s[1]-1:
        return u*array[j][i]+u1*array[j][i+1]
    return t1*u1*array[j][i]+t*u1*array[j][i+1]+\
           t*u*array[j+1][i+1]+t1*u*array[j+1][i]
```

Voici la suppression des non-maxima :

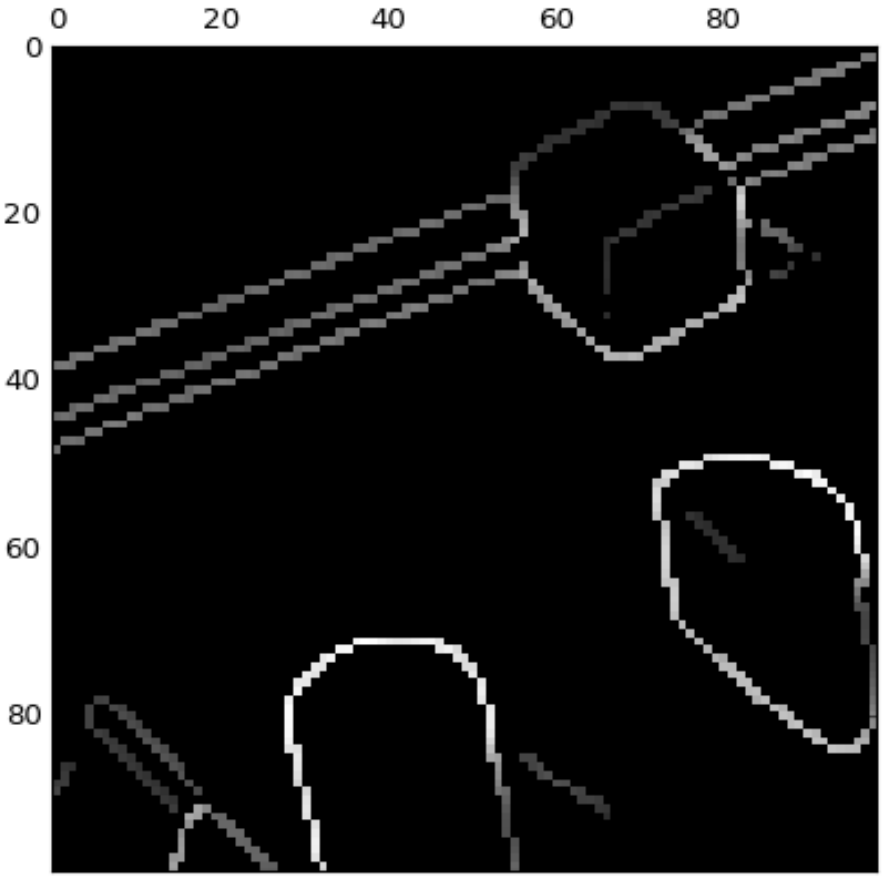
```
for i in range(1,s[1]-1):
    for j in range(1,s[0]-1):
        if G[j][i]!=0:
            cos = math.cos(theta[j][i])
            sin = math.sin(theta[j][i])
            g1 = interpolation(G,i+cos,j+sin)
            g2 = interpolation(G,i-cos,j-sin)
            if (G[j][i]<g1) or (G[j][i]<g2):
                Gmax[j][i] = 0.0

figure(figsize=(6,6))
imshow(Gmax,cmap=cm.gray)
```

Les pixels des bords ne sont pas traités (il faudrait faire pour ces points une extrapolation). On remarque que le cosinus et le sinus de l'angle auraient pu être calculés au départ et stockés dans deux matrices.



Le détail de l'image :

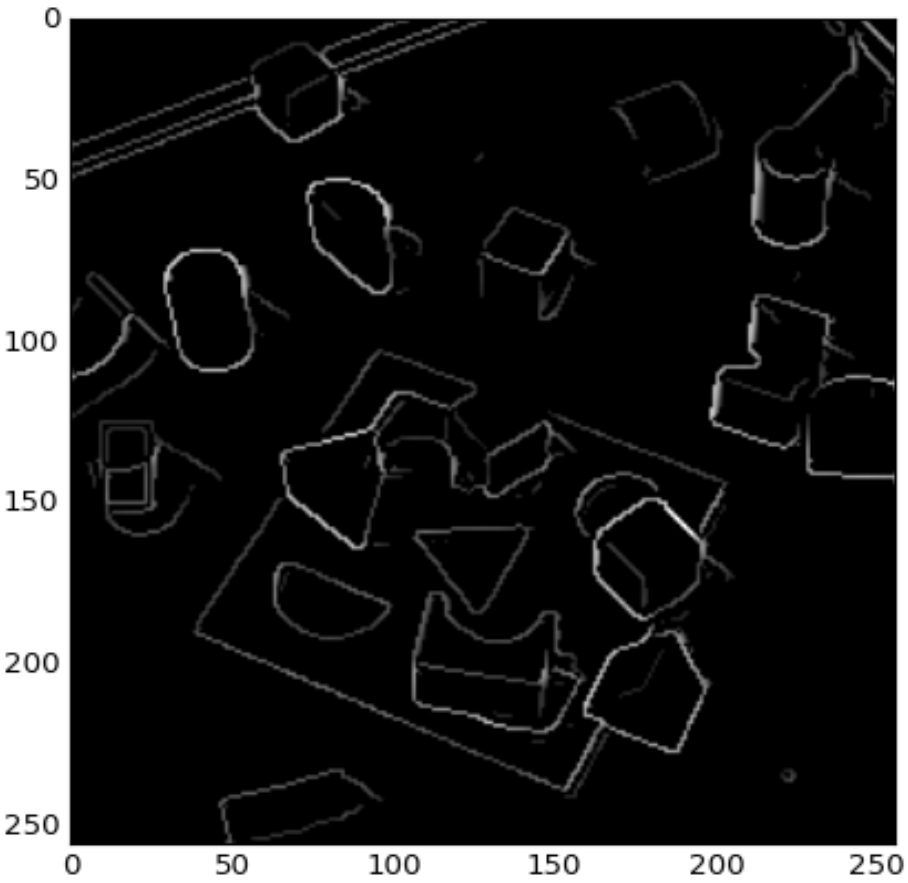
```
figure(figsize=(4,4))
matshow(Gmax[1:100,1:100],cmap=cm.gray)
```



Une autre méthode consiste à discrétiser l'angle pour lui attribuer les valeurs 0,45,90,135 ou 180 degrés. Par exemple, si l'angle de départ est compris entre -22.5 et 22.5 degrés, on lui attribue la valeur 0. Si l'angle vaut 0, on compare le pixel central avec les deux pixels situés horizontalement, si l'angle est 45 on compare avec les pixels sur la diagonale, etc. Cette méthode est plus rapide car elle évite les calculs de l'interpolation linéaire. Elle se réduit à effectuer des tests.

```
Gmax_2 = G.copy()
pi = math.pi
a = numpy.zeros(4)
a[0] = pi/8
for k in range(1,4):
    a[k] = a[k-1]+pi/4
for j in range(1,s[0]-1):
    for i in range(1,s[1]-1):
        if G[j][i]!=0:
            b = theta[j][i]
            if b>0:
                if (b<a[0]) or (b>a[3]):
                    g1 = G[j][i+1]
```

```
                    g2 = G[j][i+1]
                elif (b<a[1]):
                    g1 = G[j+1][i+1]
                    g2 = G[j-1][i-1]
                elif (b<a[2]):
                    g1 = G[j+1][i]
                    g2 = G[j-1][i]
                else:
                    g1 = G[j+1][i-1]
                    g2 = G[j-1][i+1]
            elif b<0:
                if (b<-a[3]):
                    g1 = G[j][i+1]
                    g2 = G[j][i-1]
                elif (b<-a[2]):
                    g1 = G[j-1][i-1]
                    g2 = G[j+1][i+1]
                elif (b<-a[1]):
                    g1 = G[j-1][i]
                    g2 = G[j+1][i]
                elif (b<-a[0]):
                    g1 = G[j-1][i+1]
                    g2 = G[j+1][i-1]
                else:
                    g1 = G[j][i+1]
                    g2 = G[j][i-1]
            if (G[j][i]<g1) or (G[j][i]<g2):
                Gmax_2[j][i] = 0.0


figure(figsize=(6,6))
imshow(Gmax_2,cmap=cm.gray)
```



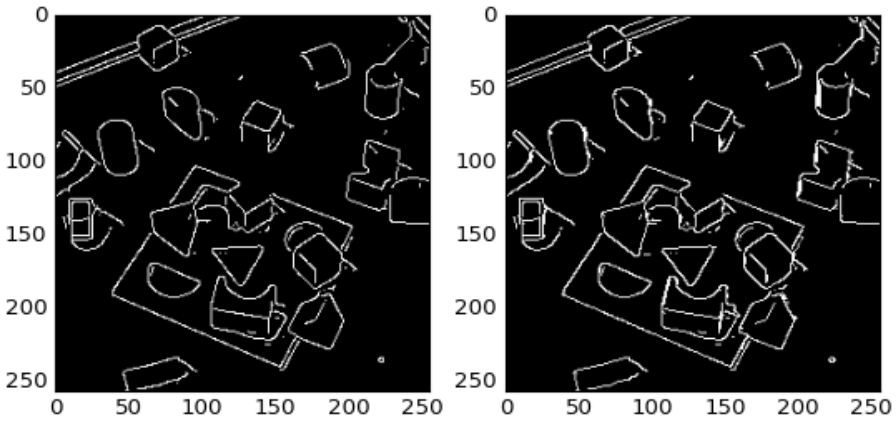
Le résultat est très voisin de celui obtenu avec l'interpolation bilinéaire.

4. Seuillage

La dernière étape consiste à binariser l'image avec un seuil :

```
Gfinal = Gmax.copy()
Gfinal_2 = Gmax_2.copy()
seuil = 0.2
for j in range(s[0]):
    for i in range(s[1]):
        if Gfinal[j][i]<seuil:
            Gfinal[j][i] = 0.0
        else:
            Gfinal[j][i] = 255.0
        if Gfinal_2[j][i]<seuil:
            Gfinal_2[j][i] = 0.0
        else:
            Gfinal_2[j][i] = 255.0
```

```
figure(figsize=(10,6))
f,(p1,p2)=subplots(ncols=2)
p1.imshow(Gfinal,cmap=cm.gray)
p2.imshow(Gfinal_2,cmap=cm.gray)
```



L'image de gauche est obtenue par interpolation bilinéaire, celle de droite par discrétisation de l'angle.

Références

[1] M. Nixon, A. Aguado, *Feature extraction and image processing*, (Academic Press, 2008)