IV. Détection de contours

```
43  %---
44  % mise en évidence du contour
45  % somme sur les colonnes
46  sumP=sum(JPs); sumS=sum(JSs); sumR=sum(JRs);
47  % détection du contour
48  dimP = (size(JPs))(1);
49  dimS = (size(JSs))(1);
50  dimR = (size(JRs))(1);
51  gooP = sumP(dimP/2:dimP/2+1);
52  gooS = sumS(dimS/2:dimS/2+1);
53  gooR = sumR((dimR+1)/2);
54
55  {'----------------------' ;
56   'Détection du contour...' ;
57   '----------------------' ;
58   ['Prewitt : ', num2str(dimP), '   ', num2str(gooP)] ;
59   ['Sobel   : ', num2str(dimS), '   ', num2str(gooS)] ;
60   ['Roberts : ', num2str(dimR), '   ', num2str(gooR)] ;
61   '------------------------' }
62  % => probabilités de détection
63  {'-----------------------------' ;
64   '=> Probabilités de détection...' ;
65   '-----------------------------' ;
66   ['Prewitt : ', num2str(sum(gooP)/(2*dimP))] ;
67   ['Sobel   : ', num2str(sum(gooS)/(2*dimS))] ;
68   ['Roberts : ', num2str(sum(gooR)/dimR)] ;
69   '-----------------------------' }
```

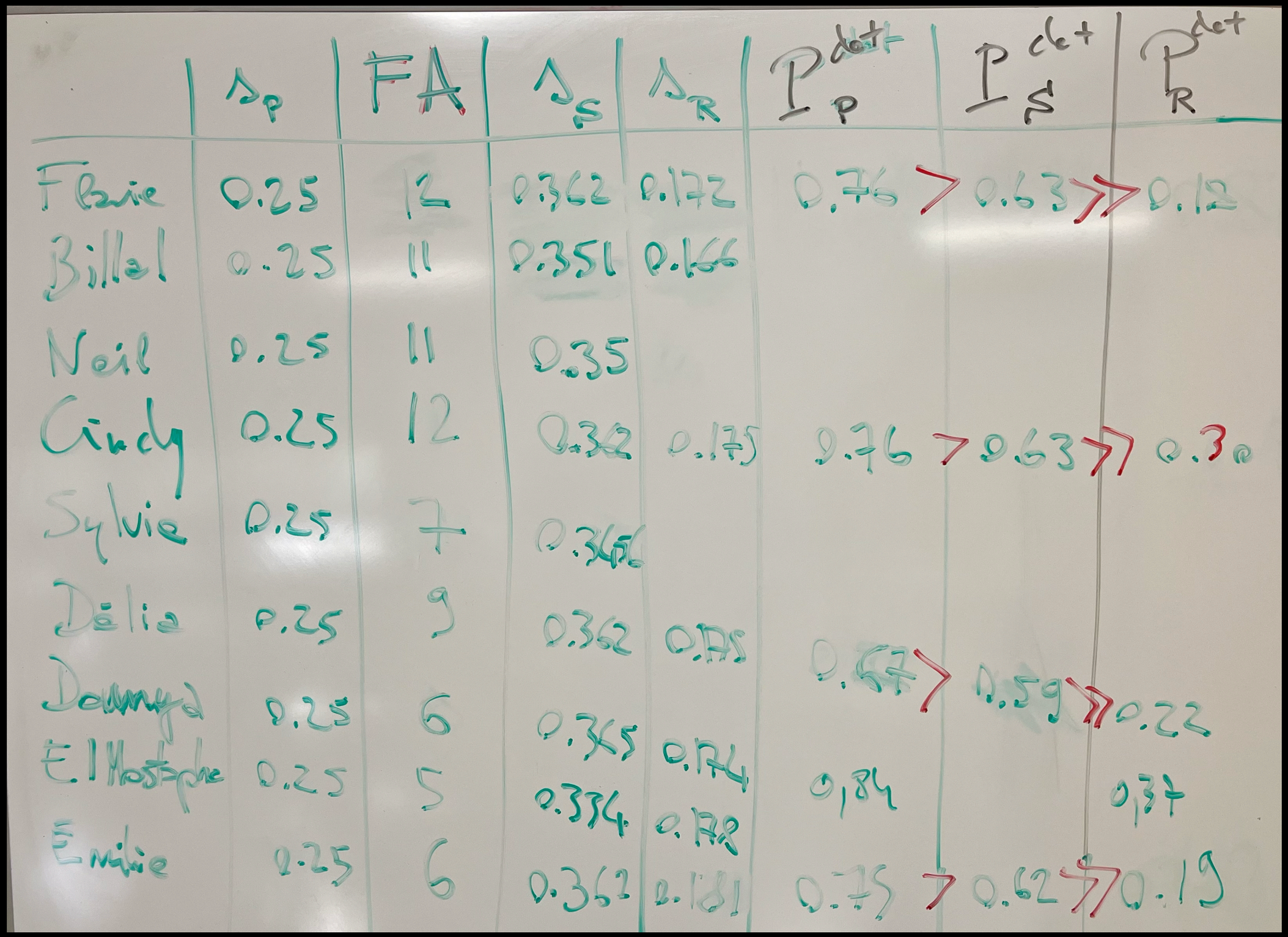
```
dummy=size(JPs), dimP = dummy(1);
dummy=size(JSs), dimS = dummy(1);
dummy=size(JRs), dimR = dummy(1);
```

```
ans =
{
  [1,1] = ----------------------
  [2,1] = Détection du contour...
  [3,1] = ----------------------
  [4,1] = Prewitt : 126  107   97
  [5,1] = Sobel   : 126  95  83
  [6,1] = Roberts : 127  36
  [7,1] = ------------------------
}

ans =
{
  [1,1] = -----------------------------
  [2,1] = => Probabilités de détection...
  [3,1] = -----------------------------
  [4,1] = Prewitt : 0.80952
  [5,1] = Sobel   : 0.70635
  [6,1] = Roberts : 0.28346
  [7,1] = -----------------------------
}
```

- **Conclusion : Prewitt > Sobel >> Roberts**
- **Mais : Roberts détecte un « vrai » bord, de 1 pixel de large (pas 2)…**

IV. Détection de contours



	Δ_P	FA	Δ_S	Δ_R	P_P^{det}	P_S^{det}	P_R^{det}
Flavie	0.25	12	0.362	0.172	0.76	> 0.63	>> 0.12
Billal	0.25	11	0.351	0.166			
Neil	0.25	11	0.35				
Cindy	0.25	12	0.32	0.175	0.76	> 0.63	>> 0.30
Sylvie	0.25	7	0.365				
Délia	0.25	9	0.362	0.215			
Dounyd	0.25	6	0.365	0.174	0.67	> 0.59	>> 0.22
El Mostapha	0.25	5	0.334	0.18	0,84		0,37
Emilie	0.25	6	0.362	0.181	0.75	> 0.62	>> 0.19

IV. Détection de contours

- **EXERCICE 3 (suite) : Calculer les P_{fa} correspondantes « à la main ». Puis coder ce calcul de P_{fa} de manière à le rendre automatique.**

Théoriquement, on doit avoir :

$$P_{\text{fa}}^{P} = P_{\text{fa}}^{S} = 9/(126 \times 124) \simeq 5.7604\,10^{-4}$$

$$P_{\text{fa}}^{R} = 9/(127 \times 126) \simeq 5.6243\,10^{-4}$$

Et en effet :

```
71  %---
72  % probabilité de fausse alarme
73  FA_ga_P = sum(sumP(1:dimP/2-1));
74  FA_dr_P = sum(sumP(dimP/2+2:dimP));
75  FA_ga_S = sum(sumS(1:dimS/2-1));
76  FA_dr_S = sum(sumS(dimS/2+2:dimS));
77  FA_ga_R = sum(sumR(1:(dimR+1)/2-1));
78  FA_dr_R = sum(sumR((dimR+1)/2+1:dimR));
79  {'--------------------------' ;
80   'Probabilité de fausse alarme' ;
81   '--------------------------' ;
82   ['Prewitt : ', num2str((FA_ga_P+FA_dr_P)/((dimP-2)*dimP))] ;
83   ['Sobel   : ', num2str((FA_ga_S+FA_dr_S)/((dimS-2)*dimS))] ;
84   ['Roberts : ', num2str((FA_ga_R+FA_dr_R)/((dimR-1)*dimR))] ;
85    '--------------------------' }
```

```
ans =
{
  [1,1] = --------------------------
  [2,1] = Probabilité de fausse alarme
  [3,1] = --------------------------
  [4,1] = Prewitt : 0.00057604
  [5,1] = Sobel   : 0.00057604
  [6,1] = Roberts : 0.00056243
  [7,1] = --------------------------
}
```

IV. Détection de contours

- **EXERCICE 3bis : Partir de P_{fa} ≈ 1%, en déduire le nombre de fausses alarmes (N_{fa}) correspondant (pour chaque cas : Prewitt, Sobel, Roberts), en déduire alors (pour chaque cas aussi) le seuil nécessaire (en comptant automatiquement N_{fa}), puis calculer les $P_{dét.}$ résultantes à partir des images de contour unique seuillées correspondantes.**

P_{fa}=1% => N_{fa} = 156 pour Prewitt et Sobel (1% de 124x126) et N_{fa} = 160 pour Roberts (1% de 126x127).

IV. Détection de contours

- **De manière théorique, on peut prévoir la valeur du seuil qui donne une certaine probabilité de fausse alarme :**

$$P_{\text{f.a.}} = \exp\left(-\frac{s^2}{2\sigma^2}\right) \Rightarrow s = \sqrt{-2\,\sigma^2 \ln\left(P_{\text{f.a.}}\right)}$$

Avec : variance = somme des carrés des coeff. du filtre x variance du bruit Gaussien présent dans l'image filtrée.

Et : P_{fa} = nombre de fausses alarmes/nombre total de pixels qui ne sont pas du contour.

	6	0.006	(124x126)	0.00057604
Ici : variance = 12 x 0.001 = 0.012 ; Et : P_{fa} = 9 / (124x126) ≈ 0.00057604				
	2	0.002	(126x127)	0.00056243

D'où : $s_{Prewitt}$ = sqrt(-2*ln(.00057604)*.006) ≈ 0.299 (à comp. au 0.250 utilisé)
s_{Sobel} = sqrt(-2*ln(.00057604)*.012) ≈ 0.423 (à comp. au 0.362 utilisé)
$s_{Roberts}$= sqrt(-2*ln(.00056243)*.002) ≈ 0.173 (à comp. au 0.175 utilisé)

IV. Détection de contours

6- REMARQUE FINALE

- **Il existe pléthore de méthodes permettant la détection de contour. La fonction edge en intègre plusieurs, dont l'utilisation des paires de filtres Prewitt, Sobel et Roberts que nous avons vue précédemment, avec même quelques options supplémentaires (telle que « thinning » qui permet à Prewitt ou Sobel de produire des contours d'un seul pixel de large).**

La fonction edge intègre également d'autres méthodes intéressantes, telle que par exemple celle de *Canny* faisant usage de deux seuils, ce qui doit permettre de s'affranchir plus facilement du bruit.

```
>> help edge
'edge' is a function from the file /Users/marcel/Library/Application Suppor
t/Octave.app/4.4.1/pkg/image-2.10.0/edge.m

 -- Function File: [BW, THRESH] = edge (IM, METHOD, ...)
     Find edges using various methods.

     The image IM must be 2 dimensional and grayscale.  The METHOD must
     be a string with the string name.  The other input arguments are
     dependent on METHOD.

     BW is a binary image with the identified edges.  THRESH is the
     threshold value used to identify those edges.  Note that THRESH is
     used on a filtered image and not directly on IM.

     See also: fspecial.
```

IV. Détection de contours

```
-- Function File: edge (IM, "Prew
    Find edges using the Prewitt

    This method is the same as Ki
    gradient is used.

-- Function File: edge (IM, "Robe
-- Function File: edge (IM, "Robe
-- Function File: edge (IM, "Robe
    Find edges using the Roberts

    This method is similar to Kir
    gradient is used, and the def
    sqrt(1.5).  In addition, ther
    argument.

-- Function File: edge (IM, "Sobe
    Find edges using the Sobel ap

    This method is the same as Ki
    gradient is used.
```

```
-- Function File: edge (IM, "Kirsch")
-- Function File: edge (IM, "Kirsch", THRESH)
-- Function File: edge (IM, "Kirsch", THRESH, DIRECTION)
-- Function File: edge (IM, "Kirsch", THRESH, DIRECTION, THINNING)
    Find edges using the Kirsch approximation to the derivatives.

    Edge points are defined as points where the length of the gradient
    exceeds a threshold THRESH.

    THRESH is the threshold used and defaults to twice the square root
    of the mean of the gradient squared of IM.

    DIRECTION is the direction of which the gradient is approximated
    and can be "vertical", "horizontal", or "both" (default).

    THINNING can be the string "thinning" (default) or "nothinning".
    This controls if a simple thinning procedure is applied to the edge
    image such that edge points also need to have a larger gradient
    than their neighbours.  The resulting "thinned" edges are only one
    pixel wide.

-- Function File: edge (IM, "Lindeberg")
-- Function File: edge (IM, "Lindeberg", SIGMA)
    Find edges using the the differential geometric single-scale edge
    detector by Tony Lindeberg.

    SIGMA is the scale (spread of Gaussian filter) at which the edges
    are computed.  Defaults to '2'.

    This method does not use a threshold value and therefore does not
    return one.
```

IV. Détection de contours

```
-- Function File: edge (IM, "Canny")
-- Function File: edge (IM, "Canny", THRESH)
-- Function File: edge (IM, "Canny", THRESH, SIGMA)
    Find edges using the Canny method.

THRESH is two element vector for the hysteresis thresholding.  The
lower and higher threshold values are the first and second elements
respectively.  If it is a scalar value, the lower value is set to
'0.4 * THRESH'.

SIGMA is the standard deviation to be used on the Gaussian filter
that is used to smooth the input image prior to estimating
gradients.  Defaults to 'sqrt (2)'.
```

IV. Détection de contours

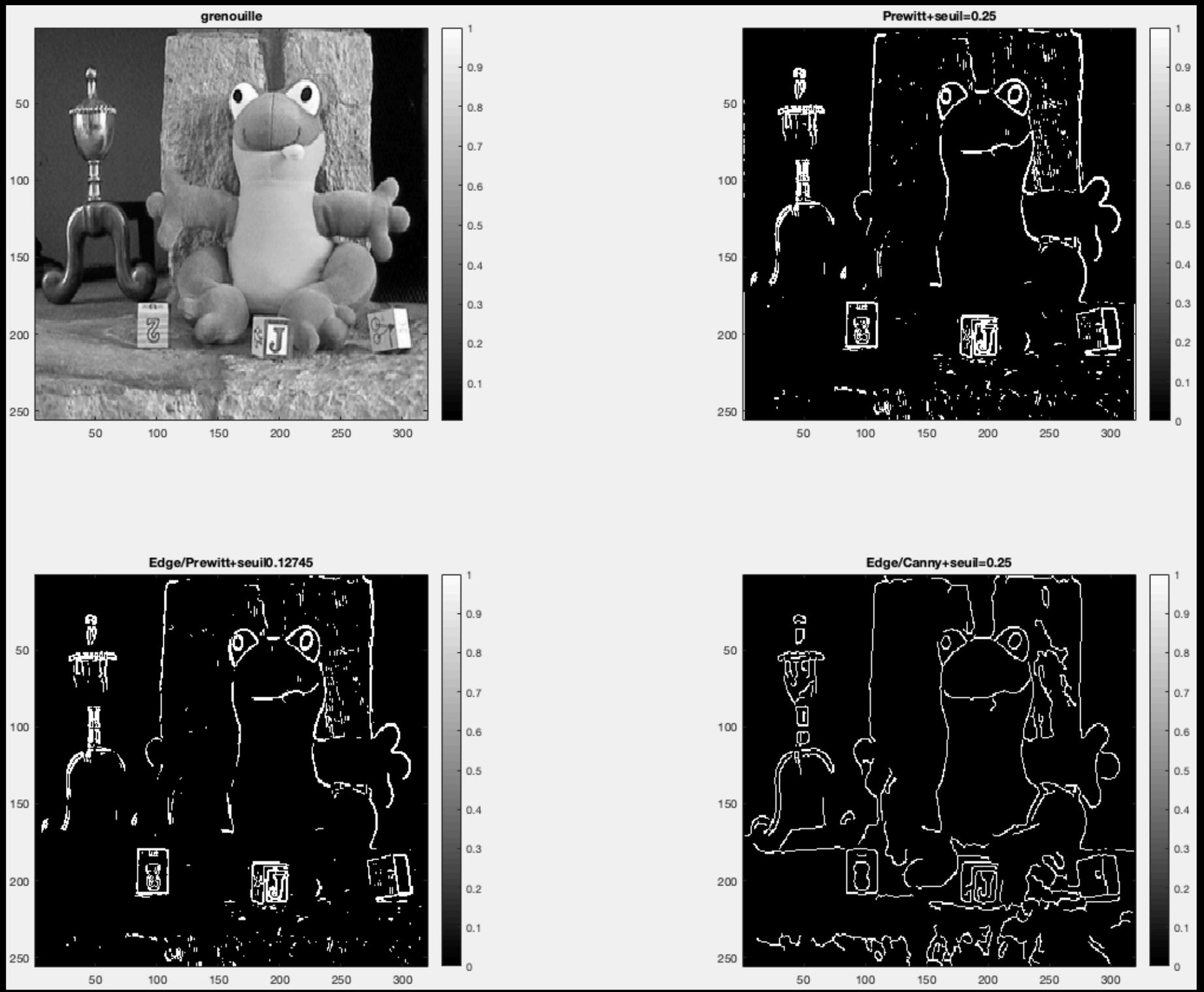
- **EXERCICE 4 : Reprendre l'image plus complexe de la grenouille. Lui appliquer le filtre de Prewitt avec un seuil de, par exemple, 0.25. Penser à ramener les valeurs de l'image de contours unique entre 0 et 1. Comparer le résultat en utilisant l'option 'prewitt' de la fonction edge (seuil par défaut, pas de « thinning »), puis en utilisant la méthode de Canny (en préférant un seuil à déterminer plutôt que celui par défaut afin de pouvoir rendre plus comparables entre eux les résultats obtenus).**

IV. Détection de contours

```
clear
close all
%pkg load image

%---
% image de la grenouille non-bruitée
img='/Users/marcel/Documents/MATLAB/GBM/0-images/frog.jpg';
frog=imread(img);
I=rgb2gray(frog);
J=double(I)/255.;
% Prewitt
Ph=fspecial('prewitt'); Pv=-Ph';
IPh=filter2(Ph,J,'same'); IPv=filter2(Pv,J,'same');
IPvh=sqrt(IPv.^2+IPh.^2);
IPvh=IPvh-min(min(IPvh)); IPvh=IPvh/max(max(IPvh));
seuilP=.25; IPs=IPvh>seuilP;
% Edge/Prewitt
%'seuil par défaut utilisé par Edge/Prewitt', 2*sqrt(mean(mean(IPvh.^2)))
[JE, seuilP1]=edge(J, 'prewitt', 'nothinning');
'seuil par défaut utilisé par Edge/Prewitt (sans bruit)', seuilP1
% Edge/Canny
seuilC=0.25;
JC=edge(J, 'canny', seuilC);
% figure
figure(1), colormap(gray)
subplot(2,2,1), imagesc(J), title('grenouille'), colorbar, axis('square')
subplot(2,2,2), imagesc(IPs), colorbar, axis('square')
   title(['Prewitt+seuil=',num2str(seuilP)])
subplot(2,2,3), imagesc(JE), colorbar, axis('square')
   title(['Edge/Prewitt+seuil', num2str(seuilP1)])
subplot(2,2,4), imagesc(JC), colorbar, axis('square')
   title(['Edge/Canny+seuil=',num2str(seuilC)])
```

IV. Détection de contours



IV. Détection de contours

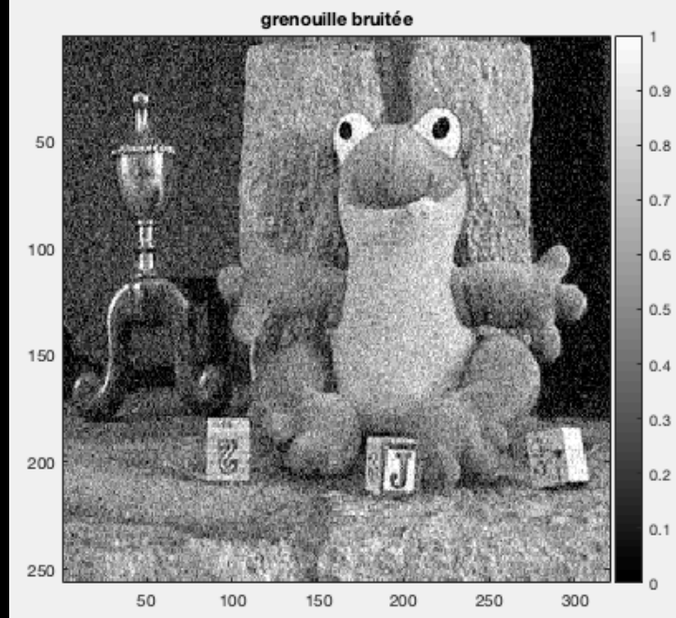
- **EXERCICE 4bis : Même chose avec une image significativement bruitée (bruit gaussien additif, variance de 0.01).**

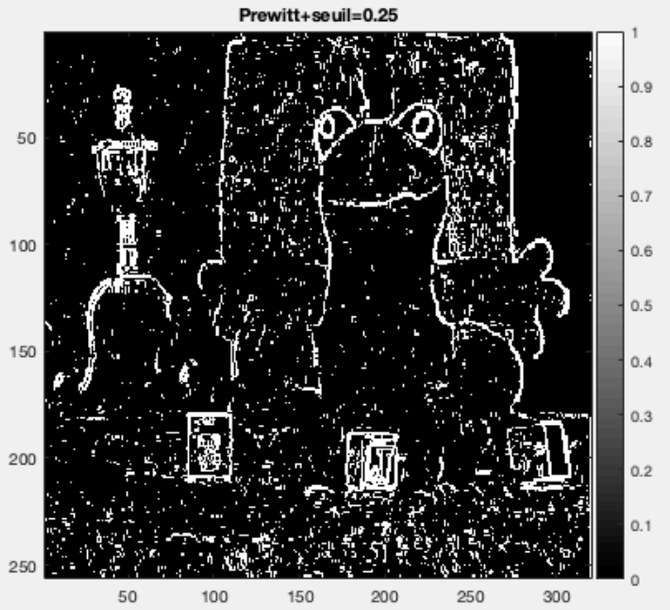
(Bien écrêter les valeurs de l'image bruitée qui pourraient descendre en-dessous de 0 ou dépasser 1, pour être compatible avec le fonctionnement de edge qui réclame des images entre 0 et 1.)

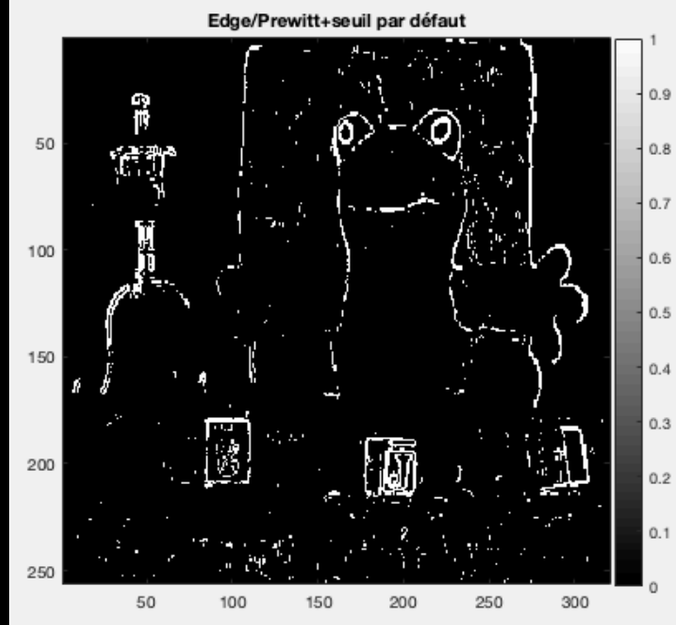
IV. Détection de contours

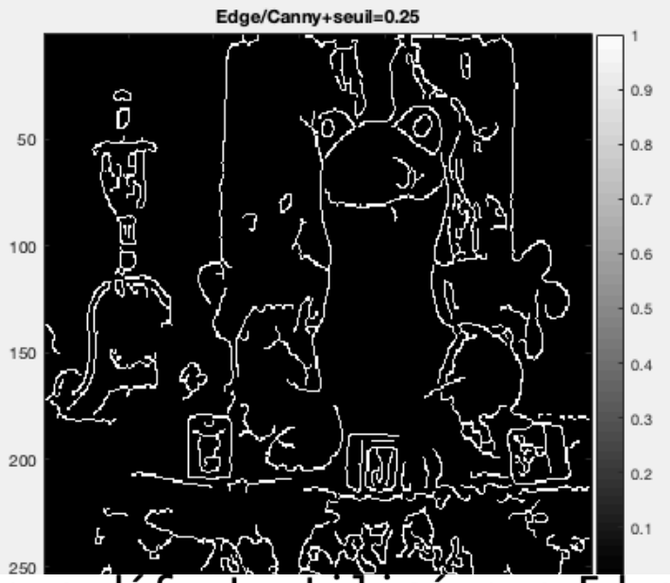
```
%---
% image de la grenouille bruitée
J=imnoise(J, 'gaussian', 0., .01);
% écrétage de l'image entre 0 et 1
idx=find(J<0); J(idx)=0.;
idx=find(J>1); J(idx)=1.;
% Prewitt
Ph=fspecial('prewitt'); Pv=-Ph';
IPh=filter2(Ph,J,'same'); IPv=filter2(Pv,J,'same');
IPvh=sqrt(IPv.^2+IPh.^2);
IPvh=IPvh-min(min(IPvh)); IPvh=IPvh/max(max(IPvh));
seuilP=.25; IPs=IPvh>seuilP;
% Edge/Prewitt
%'seuil par défaut utilisé par Edge/Prewitt', 2*sqrt(mean(mean(IPvh.^2)))
[JE, seuilP2]=edge(J, 'prewitt', 'nothinning');
'seuil par défaut utilisé par Edge/Prewitt (avec bruit)', seuilP2
% Edge/Canny
JC=edge(J, 'canny', seuilC);
% figure
figure(2), colormap(gray)
subplot(2,2,1), imagesc(J), title('grenouille bruitée'), colorbar, axis('square')
subplot(2,2,2), imagesc(IPs), colorbar, axis('square')
   title(['Prewitt+seuil=',num2str(seuilP)])
subplot(2,2,3), imagesc(JE), colorbar, axis('square')
   title('Edge/Prewitt+seuil par défaut')
subplot(2,2,4), imagesc(JC), colorbar, axis('square')
   title(['Edge/Canny+seuil=',num2str(seuilC)])
```

IV. Détection de contours










```
ans = seuil par défaut utilisé par Edge/Prewitt (sans bruit)
seuilP1 = 0.1276
ans = seuil par défaut utilisé par Edge/Prewitt (avec bruit)
seuilP2 = 0.1685
```