

IV. Détection de contours

\	nb de FA	seuil Prewitt	seuil Sobel	seuil Roberts	Pdét Prewitt	Pdét Sobel	Pdét Roberts
Jérémy Julien	8	0,24	0,37	0,147	0,760	0,619	0,173
Alexis	9	0,25	0,333	0,14	0,79	0,67	0,15
Inès	8	0,27	0,38992	0,175	0,647	0,539	0,299
Yumna	2	0,28	0,39	0,185	0,63	0,55	0,23
Yasmine	2	0,28	0,42	0,17	0,560	0,413	0,095
Annabelle, Myriam	15	0,26	0,3733	0,14453	0,67	0,57	0,15
Nathan, Soufyan,	18	0,29	0,33	0,19	0,75	...	0,07
Bouchra	6	0,33	0,376	0,145	0,76	0,62	0,29

IV. Détection de contours

- **EXERCICE 3bis** : Partir de $P_{fa} \approx 1\%$, 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 $124 \times 126 = 156.24$),
et $N_{fa} = 160$ pour Roberts (1% de $126 \times 127 = 160.02$).

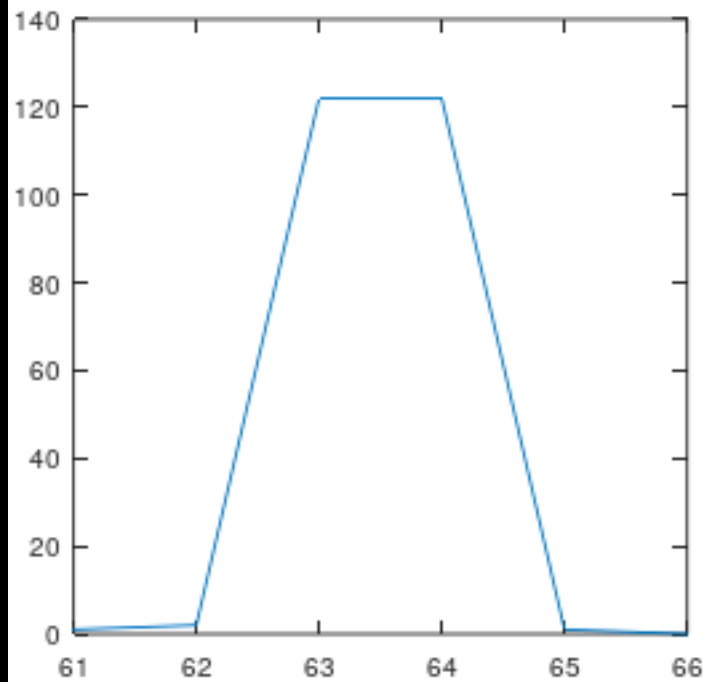
À partir de là, il faut déterminer les seuils qui permettent ces nombres de fausses alarmes (N_{fa}), puis calculer les détections et en déduire les probabilités correspondantes ($P_{dét.}$).

IV. Détection de contours

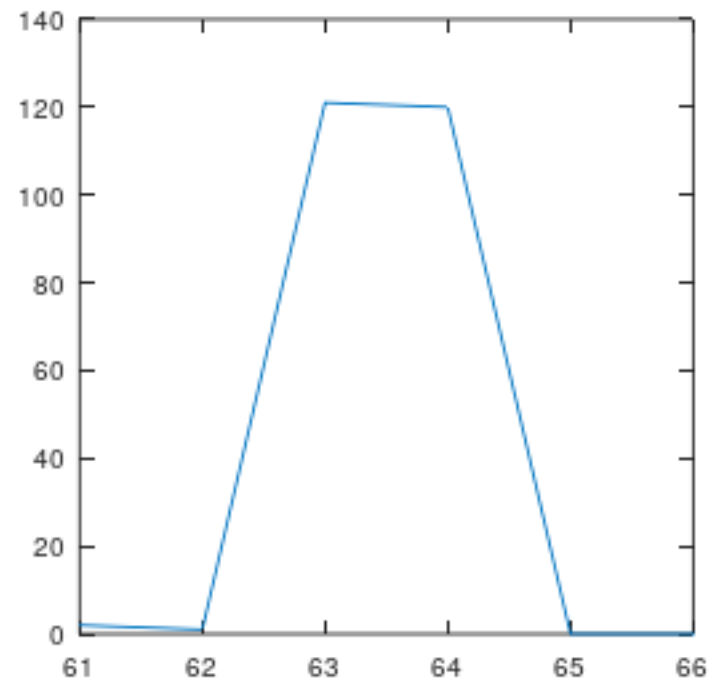
```
1 clear
2 close all
3 pkg load image
4
5 %---
6 % load image from disk
7 load image_bruit, whos J
8 %---
9 % filtrages
10 % Prewitt
11 Ph=fspecial('prewitt'); Pv=-Ph'; JP=filter2(Pv,J, 'valid');
12 % Sobel
13 Sh=fspecial('sobel'); Sv=-Sh'; JS=filter2(Sv,J, 'valid');
14 % Roberts
15 Ra=[1 0;0 -1]; Rb=rot90(Ra,-1);
16 JRa=filter2(Ra,J, 'valid'); JRb=filter2(Rb,J, 'valid');
17 JR=sqrt(JRa.*JRa+JRb.*JRb);
18 %---
19 % seuillages
20 seuilP=.179 ; JPs=JP>seuilP;
21 seuilS=.2532; JSs=JS>seuilS;
22 seuilR=.1368; JRs=JR>seuilR;
23 %---
24 % mise en évidence du contour
25 % somme sur les colonnes
26 sumP=sum(JPs); sumS=sum(JSs); sumR=sum(JRs);
27 % figure sur les colonnes de contours vs. les colonnes où résident les FA
28 figure(1)
29 subplot(1,3,1), plot(sumP), axis('square'), xlim([61 66]), title(['Prewitt, s=',num2str(seuilP)])
30 subplot(1,3,2), plot(sumS), axis('square'), xlim([61 66]), title(['Sobel, s=',num2str(seuilS)])
31 subplot(1,3,3), plot(sumR), axis('square'), xlim([61 66]), title(['Roberts, s=',num2str(seuilR)])
32 % on déduit de l'analyse de ces courbes (sumP, sumS, sumR) que les colonnes
33 % où résident les contours sont les n.63 et 64 pour Prewitt et Sobel, et la n.64
34 % pour Roberts...
```

IV. Détection de contours

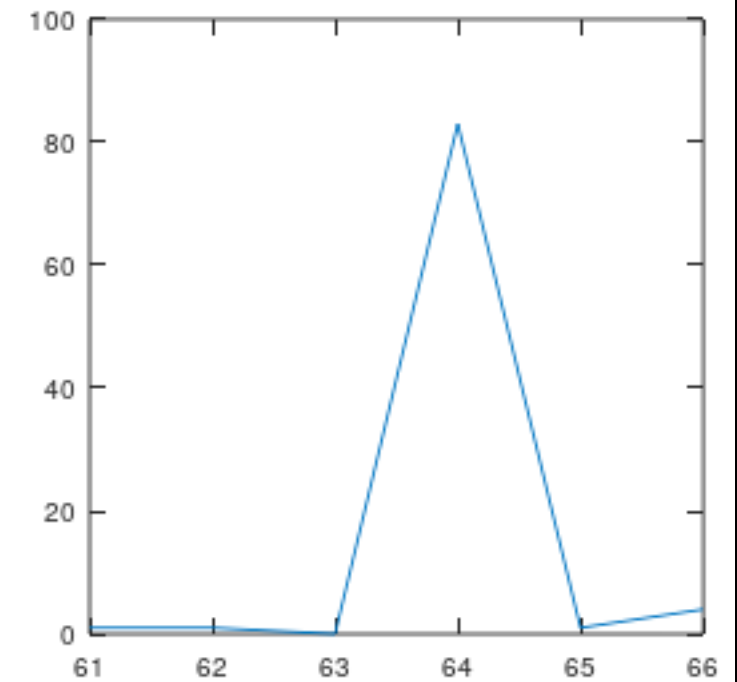
Prewitt, $s=0.179$



Sobel, $s=0.2532$



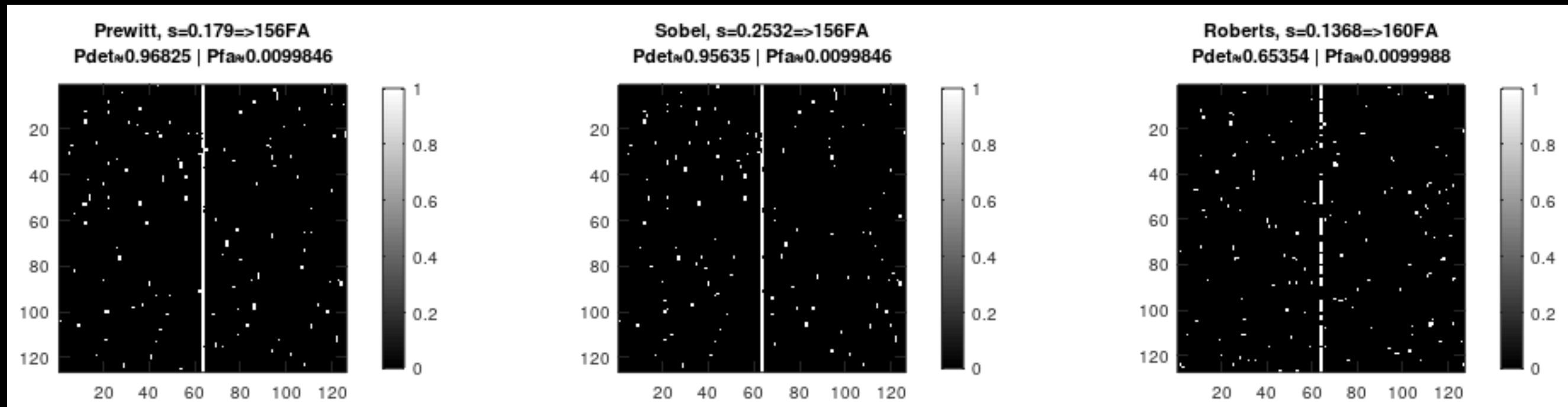
Roberts, $s=0.1368$



IV. Détection de contours

```
35 %---
36 % nombre de détections
37 DET_P=sum(sumP(63:64));
38 DET_S=sum(sumS(63:64));
39 DET_R=sum(sumR(64));
40 % => probabilités de détection
41 Pdet_P=DET_P/(2*126);
42 Pdet_S=DET_S/(2*126);
43 Pdet_R=DET_R/127;
44 %---
45 % nombre de fausses alarmes
46 FA_P=sum(sumP(1:62))+sum(sumP(65:end));
47 FA_S=sum(sumS(1:62))+sum(sumS(65:end));
48 FA_R=sum(sumR(1:63))+sum(sumR(65:end));
49 % => probabilité de fausse alarme
50 Pfa_P=FA_P/(124*126);
51 Pfa_S=FA_S/(124*126);
52 Pfa_R=FA_R/(126*127);
53 %---
54 % figure finale
55 figure(2)
56 subplot(1,3,1), colormap(gray), imagesc(JPs), colorbar, axis('square')
57 title(['Prewitt, s=',num2str(seuilP),'=>',num2str(FA_P),'FA'];['Pdet≈',num2str(Pdet_P),' | Pfa≈',num2str(Pfa_P)]))
58 subplot(1,3,2), colormap(gray), imagesc(JSs), colorbar, axis('square')
59 title(['Sobel, s=',num2str(seuilS),'=>',num2str(FA_S),'FA'];['Pdet≈',num2str(Pdet_S),' | Pfa≈',num2str(Pfa_S)]))
60 subplot(1,3,3), colormap(gray), imagesc(JRs), colorbar, axis('square')
61 title(['Roberts, s=',num2str(seuilR),'=>',num2str(FA_R),'FA'];['Pdet≈',num2str(Pdet_R),' | Pfa≈',num2str(Pfa_R)]))
```

IV. Détection de contours



IV. Détection de contours

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

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

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 = $12 \times 0.001 = 0.012$;		Et : $P_{fa} = 9 / (124 \times 126) \approx$	0.00057604
2	0.002	(126x127)	0.00056243

D'où : $S_{Prewitt} = \sqrt{-2 \ln(.00057604) \times .006} \approx 0.299$ (à comp. à 0.250)
 $S_{Sobel} = \sqrt{-2 \ln(.00057604) \times .012} \approx 0.423$ (à comp. à 0.362)
 $S_{Roberts} = \sqrt{-2 \ln(.00056243) \times .002} \approx 0.173$ (à comp. à 0.175)

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 devant permettre de s'affranchir plus facilement du bruit.

```
>> help edge
'edge' is a function from the file /Users/marcel/Library/Application Support
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, "Prewitt")
Find edges using the Prewitt

This method is the same as Kirsch
gradient is used.

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

This method is similar to Kirsch
gradient is used, and the default
sqrt(1.5). In addition, there is a
argument.

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

This method is the same as Kirsch
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 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 ' $\sqrt{2}$ '.

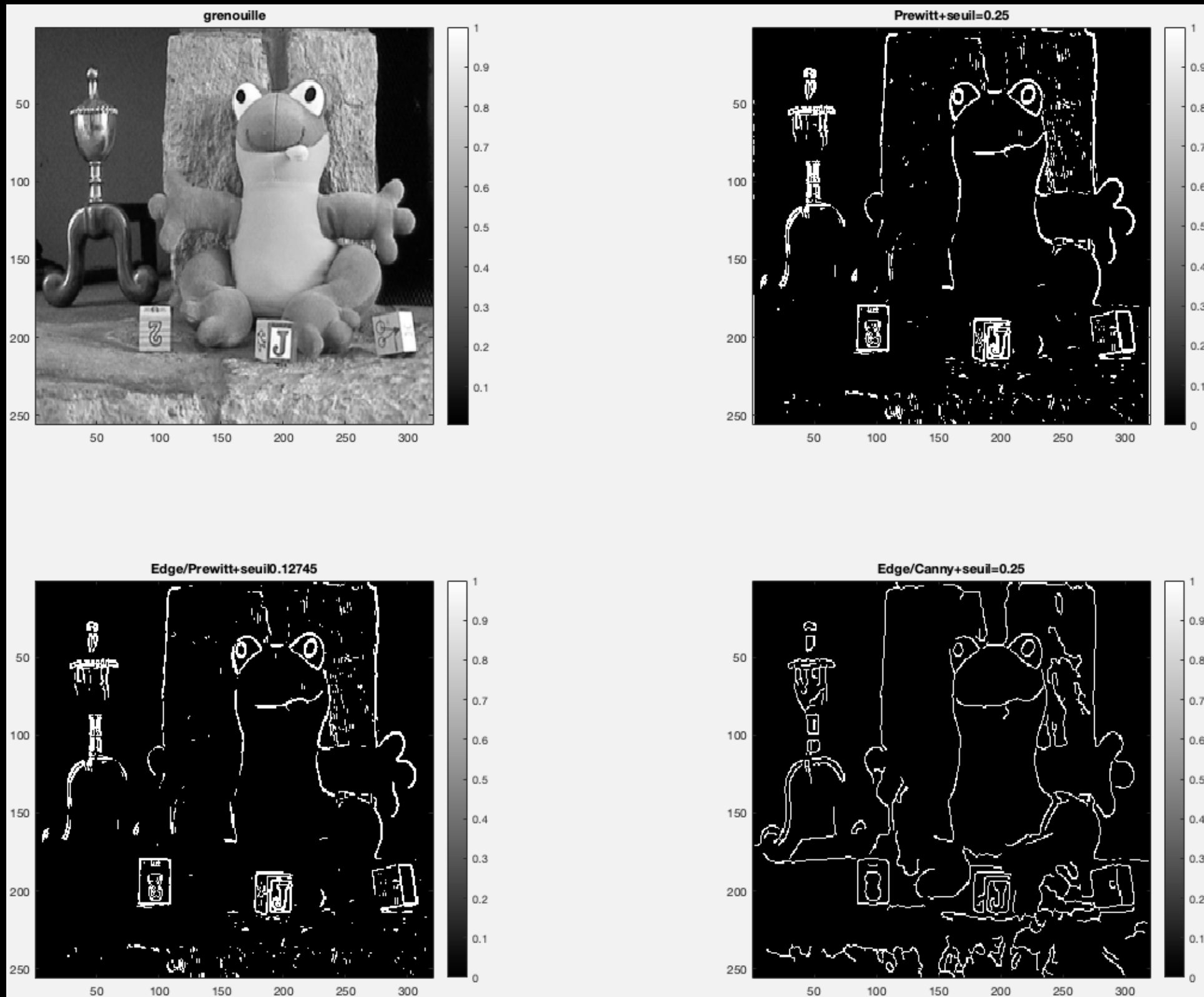
IV. Détection de contours

- **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

```
1 clear
2 close all
3 %pkg load image
4
5 %---
6 % image de la grenouille non-bruitée
7 img='/Users/marcel/Documents/MATLAB/GBM/0-images/frog.jpg';
8 frog=imread(img);
9 I=rgb2gray(frog);
10 J=double(I)/255.;
11 % Prewitt
12 Ph=fspecial('prewitt'); Pv=-Ph';
13 IPh=filter2(Ph,J,'same'); IPv=filter2(Pv,J,'same');
14 IPvh=sqrt(IPv.^2+IPh.^2);
15 IPvh=IPvh-min(min(IPvh)); IPvh=IPvh/max(max(IPvh));
16 seuilP=.25; IPs=IPvh>seuilP;
17 % Edge/Prewitt
18 %'seuil par défaut utilisé par Edge/Prewitt', 2*sqrt(mean(mean(IPvh.^2)))
19 [JE, seuilP1]=edge(J, 'prewitt', 'nothinning');
20 'seuil par défaut utilisé par Edge/Prewitt (sans bruit)', seuilP1
21 % Edge/Canny
22 seuilC=0.25;
23 JC=edge(J, 'canny', seuilC);
24 % figure
25 figure(1), colormap(gray)
26 subplot(2,2,1), imagesc(J), title('grenouille'), colorbar, axis('square')
27 subplot(2,2,2), imagesc(IPs), colorbar, axis('square')
28     title(['Prewitt+seuil=',num2str(seuilP)])
29 subplot(2,2,3), imagesc(JE), colorbar, axis('square')
30     title(['Edge/Prewitt+seuil', num2str(seuilP1)])
31 subplot(2,2,4), imagesc(JC), colorbar, axis('square')
32     title(['Edge/Canny+seuil=',num2str(seuilC)])
```

IV. Détection de contours



IV. Détection de contours

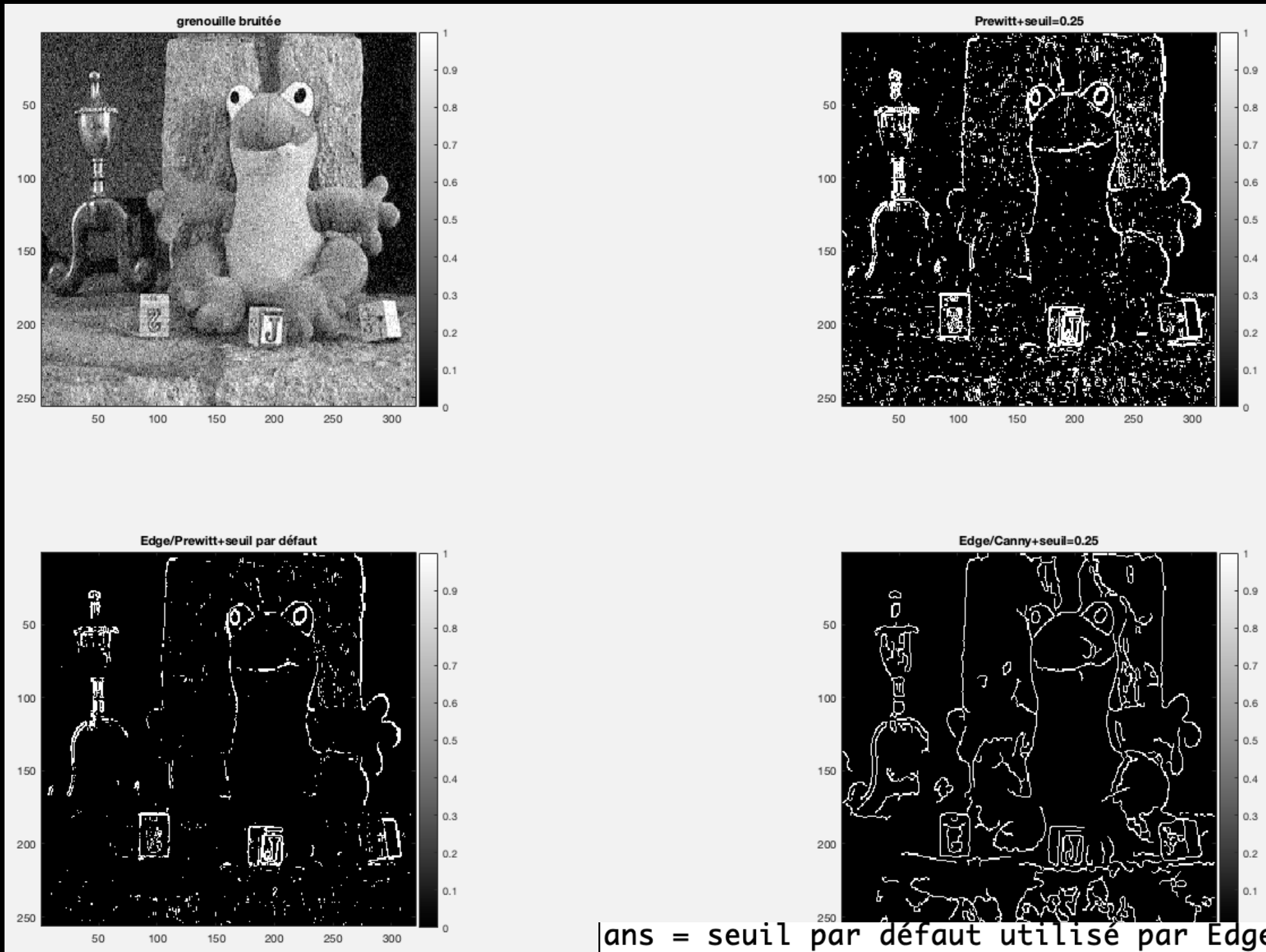
- **EXERCICE 4bis : Même chose avec une image significativement bruitée (bruit gaussien additif, variance de 0.01).**

(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

```
34 %---
35 % image de la grenouille bruitée
36 J=imnoise(J, 'gaussian', 0., .01);
37 % écrêtage de l'image entre 0 et 1
38 idx=find(J<0); J(idx)=0.;
39 idx=find(J>1); J(idx)=1.;
40 % Prewitt
41 Ph=fspecial('prewitt'); Pv=-Ph';
42 IPh=filter2(Ph,J,'same'); IPv=filter2(Pv,J,'same');
43 IPvh=sqrt(IPv.^2+IPh.^2);
44 IPvh=IPvh-min(min(IPvh)); IPvh=IPvh/max(max(IPvh));
45 seuilP=.25; IPs=IPvh>seuilP;
46 % Edge/Prewitt
47 %'seuil par défaut utilisé par Edge/Prewitt', 2*sqrt(mean(mean(IPvh.^2)))
48 [JE, seuilP2]=edge(J, 'prewitt', 'nothinning');
49 'seuil par défaut utilisé par Edge/Prewitt (avec bruit)', seuilP2
50 % Edge/Canny
51 JC=edge(J, 'canny', seuilC);
52 % figure
53 figure(2), colormap(gray)
54 subplot(2,2,1), imagesc(J), title('grenouille bruitée'), colorbar, axis('square')
55 subplot(2,2,2), imagesc(IPs), colorbar, axis('square')
56 title(['Prewitt+seuil=',num2str(seuilP)])
57 subplot(2,2,3), imagesc(JE), colorbar, axis('square')
58 title('Edge/Prewitt+seuil par défaut')
59 subplot(2,2,4), imagesc(JC), colorbar, axis('square')
60 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