TP1: CMC

Corentin Soubeiran 20/04/2020

1. Segmentation Bayésienne

L'objectif de ce TD est de réaliser une segmentation bayésienne de données. On cherche donc à classifier n réalisation alétoire observés par une variable Y en une ralisation de l'ensemble des classes Ω. Ainsi on cherche à estimer $X = (X_1, \dots X_n)$ à partir de l'observation $Y = (Y_1, \dots Y_n)$ tout en minimisant la perte, correspondant à une erreur, modélisé par la fonction:

$$\forall (i,j) \in \Omega^2, L_{0/1}(i,j) = 1_{[i \neq j]}$$

Pour minimiser cette fonction de perte on peut utiliser un estimateur MPM (mode des marginales à postériori), le problèmes se limite alors à la modélisation de la loi $p(x,y)$ régissant le lien entre la segementation X et l'observation Y. Nous comparerons ici une modélisation de chaine de Markov introduisant une "dépendance d'une donnée à son passé", les données (X_i, Y_i) sont alors dépendantes. Et le modèles des (X_i, Y_i) indépendants, prenant chaque élement indépandament aux autres.

2. Chaines de Markov

Dans ce cadre nous modélisons un processus aléatoire de n réalisations pouvant siéger dans deux classes différentes $\Omega = \omega_1, \omega_2$, avec X un processus caché $X = (X_1, \dots X_n) \in \Omega$ que l'on estime à partir de l'observation $Y = (Y_1, \dots Y_n) \in \mathbb{R}$. X étant une chaine de Markov caché ((X, Y) de Markov).

Importation des librairies:

In [1]:
```
import os
path=os.getcwd()+"/src"
os.chdir(path)
print(os.getcwd())

import numpy as np
from tools import bruit_gauss, calc_erreur
from markov_chain import *

np.random.seed(1) #1 totalement arbitraire, je fixe la graine aléatoire de manière à
 rendre mes observations consitantes aux données.
```

```
/mnt/c/Users/csoub/OneDrive/Bureau/My desktop/Scolaire/2A/P4/MA202/Tp/TP_MARKOV_1/sr
c
```

Paramètres de la simulation:

Pour la segmentation d'image nous considérons les deux états noir et blanc corrospondant aux nombres 255 et 0 (codage sur 8 bits).

Cf.README figure 1

```
mermaid
graph LR;
    0-.0,2.->255;
    0-.0,8.->0;
    255-.0,07.->0;
    255-.0,93.->255;
```

In [2]:
```
n=1000

w=np.array([0,255])
p=np.array([0.25,0.75])
A=np.array( [ [0.8, 0.2],
                            [0.07, 0.93]])
#Configuration n°1:
m1=0
m2=3
sig1=1
sig2=2
config=[[m1,m2,sig1,sig2]]
#Configuration n°2:
m1=1
m2=1
sig1=1
sig2=5
config.append([m1,m2,sig1,sig2])
#Configuration n°3:
m1=0
m2=1
sig1=1
sig2=1
config.append([m1,m2,sig1,sig2])
```

Modèle des chaines de Markov:

Création du signal et son bruitage, reconstruction du signal avec le mode des marginales à postériori:

In [3]:
```
### Génération du signal:
signal=simu_mc(n,w,p,A)

### Génération du bruit avec les différentes configurations:
signal_noisy=[]
for conf in config:
    signal_noisy.append( bruit_gauss(signal,w,conf[0],conf[2],conf[1],conf[3]) )

### Optention du nouveau signal par le mode des maginales à postériori:
new_signal=[]
i=0
for conf in config:
    new_signal.append(mpm_mc(signal_noisy[i],w,p,A,conf[0],conf[2],conf[1],conf[3]))
    i=i+1

### Calcul de l'erreur:
erreur=[]
i=0
for conf in config:
    erreur.append(calc_erreur(signal,new_signal[i]))
    print("Erreur commise avec la configuration n°{}: {:.2%}".format(i,erreur[i]))
    i=i+1
```

```
Erreur commise avec la configuration n°0: 6.70%
Erreur commise avec la configuration n°1: 8.00%
Erreur commise avec la configuration n°2: 15.40%
```

Dans l'hypothèse de stationarité, et en utilisant l'estimateur empirique des fréquences, on peut à partir du signal (non bruité) donner une estimation de la répatition de départ p et de la matrice de transition A.

In [4]:
```
p_estim,A_estim=calc_probaprio_mc(signal,w)
print('p estimé:')
print(p_estim)
print('p reel:')
print(p)
print('A estimé:')
print(A_estim)
print('A reel:')
print(A)
```

```
p estimé:
[0.28 0.72]
p reel:
[0.25 0.75]
A estimé:
[[0.82857143 0.16785714]
 [0.06666667 0.93333333]]
A reel:
[[0.8  0.2 ]
 [0.07 0.93]]
```

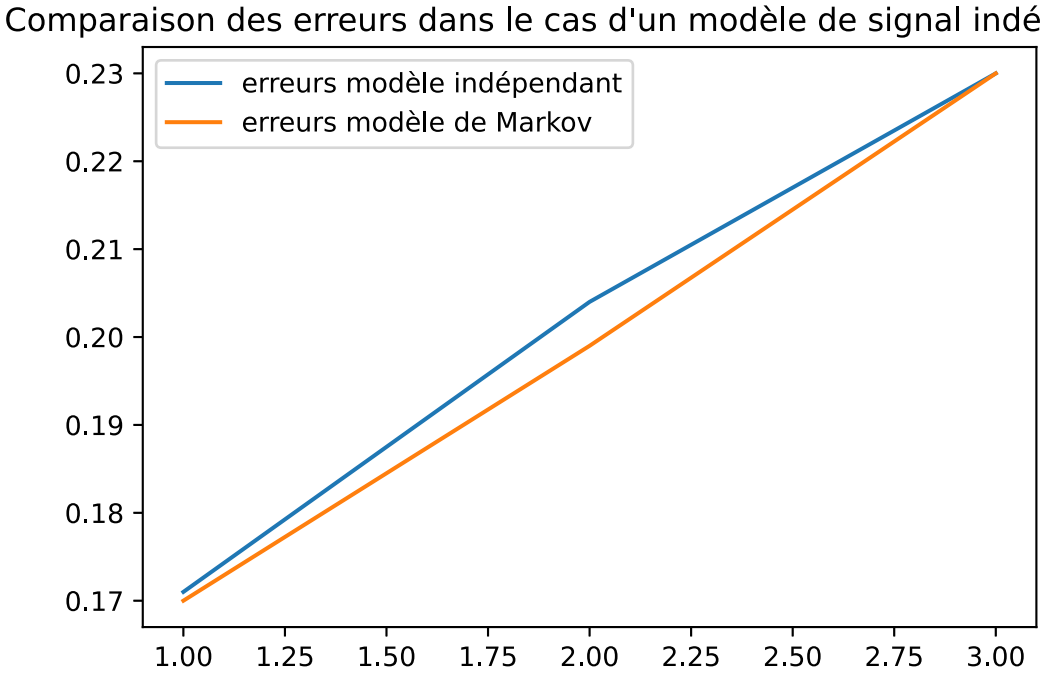
Modèle idépendant:

On va maintenant comparer le résultat de l'estimation offerte par Markov a l'estimation que l'on peut avoir avec un modèle idépendant.

génération du signal en modèle indépendant:

In [5]:
```
from gaussian_mixture import *
from matplotlib import pyplot as plt
n=1000
np.random.seed(1)
w=np.array([0,255])
p=np.array([0.25,0.75])
A=np.array( [ [0.8, 0.2],
                          [0.07, 0.93]])

##********************************
## Modèle indépendant:
##********************************
### Génération du signal:
signal=simu_gm(n,w,p)

### Génération du bruit avec les différentes configurations:
signal_noisy=[]
for conf in config:
    signal_noisy.append( bruit_gauss(signal,w,conf[0],conf[2],conf[1],conf[3]) )

### Optention des paramètres d'une chaine de Markov
p_op,A_op=calc_probaprio_mc(signal, w)

### Optention du nouveau signal par le mode des maginales à postériori pour le modèle
indep et de markov:
new_signal_indep=[]
new_signal_markov=[]
i=0
for conf in config:
    new_signal_indep.append(mpm_gm(signal_noisy[i],w,p,conf[0],conf[2],conf[1],conf[3
]))
    new_signal_markov.append(mpm_mc(signal_noisy[i],w,p_op,A_op,conf[0],conf[2],conf[
1],conf[3]))
    i=i+1

### Calcul de l'erreur:
erreur_indep=[]
erreur_markov=[]
i=0
for conf in config:
    erreur_indep.append(calc_erreur(signal,new_signal_indep[i]))
    erreur_markov.append(calc_erreur(signal,new_signal_markov[i]))
    print("Erreur commise avec la configuration n°{}: suivant le modèle indépendant:
{:.2%} , suivant le modèle de Markov {:.2%}".format(i,erreur_indep[i],erreur_markov[i
]))
    i=i+1

plt.plot([1,2,3],erreur_indep,label="erreurs modèle indépendant")
plt.plot([1,2,3],erreur_markov,label="erreurs modèle de Markov")
plt.title("Comparaison des erreurs dans le cas d'un modèle de signal indépendant")
plt.legend()
plt.show()
print(A_op)
print(p_op)
```

```
Erreur commise avec la configuration n°0: suivant le modèle indépendant:17.10% , sui
vant le modèle de Markov 17.00%
Erreur commise avec la configuration n°1: suivant le modèle indépendant:20.40% , sui
vant le modèle de Markov 19.90%
Erreur commise avec la configuration n°2: suivant le modèle indépendant:23.00% , sui
vant le modèle de Markov 23.00%
```




```
[[0.19685039 0.7992126 ]
 [0.27345845 0.72654155]]
[0.254 0.746]
```

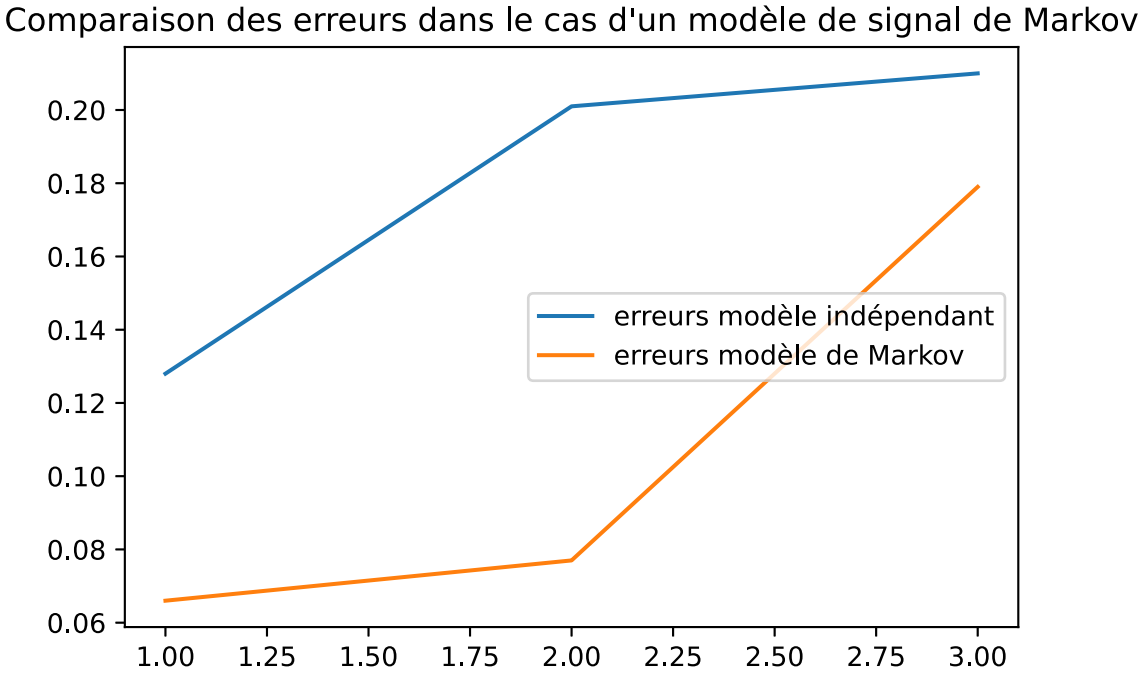
Nous remarquons que les erreurs du modèle de Markov sont légèrement meilleures que celles du modèle indépendant néamoins en faisant varier la racine aléatoire, on peut remarquer que même si globalement Markov reste meilleurs, il peut parfois être moins bon.

génération du signal en modèle de Markov:

In [6]:

```
##********************************
## Modèle de Markov:
##********************************
### Génération du signal:
signal=simu_mc(n,w,p,A)

### Génération du bruit avec les différentes configurations:
signal_noisy=[]
for conf in config:
    signal_noisy.append( bruit_gauss(signal,w,conf[0],conf[2],conf[1],conf[3]) )

### Optention des paramètres du modèle indep
p_indep_op=calc_probaprio_gm(signal, w)

### Optention du nouveau signal par le mode des maginales à postériori pour le modèle
indep et de markov:
new_signal_indep=[]
new_signal_markov=[]
i=0
for conf in config:
    new_signal_indep.append(mpm_gm(signal_noisy[i],w,p_indep_op,conf[0],conf[2],conf[
1],conf[3]))
    new_signal_markov.append(mpm_mc(signal_noisy[i],w,p,A,conf[0],conf[2],conf[1],con
f[3]))
    i=i+1

### Calcul de l'erreur:
erreur_indep=[]
erreur_markov=[]
i=0
for conf in config:
    erreur_indep.append(calc_erreur(signal,new_signal_indep[i]))
    erreur_markov.append(calc_erreur(signal,new_signal_markov[i]))
    print("Erreur commise avec la configuration n°{}: suivant le modèle indépendant:
{:.2%} , suivant le modèle de Markov {:.2%}".format(i,erreur_indep[i],erreur_markov[i
]))
    i=i+1

plt.plot([1,2,3],erreur_indep,label="erreurs modèle indépendant")
plt.plot([1,2,3],erreur_markov,label="erreurs modèle de Markov")
plt.title("Comparaison des erreurs dans le cas d'un modèle de signal de Markov")
plt.legend()
plt.show()
print(np.sum(np.array(erreur_indep)/np.array(erreur_markov))/3)
```

```
Erreur commise avec la configuration n°0: suivant le modèle indépendant:12.80% , sui
vant le modèle de Markov 6.60%
Erreur commise avec la configuration n°1: suivant le modèle indépendant:20.10% , sui
vant le modèle de Markov 7.70%
Erreur commise avec la configuration n°2: suivant le modèle indépendant:21.00% , sui
vant le modèle de Markov 17.90%
```




```
1.9076559691084831
```

On remarque cette fois ci que dans le cas d'un signal généré par un modèle de Markov, l'estimation est bien meilleurs en utilisant l'estimation avec Markov qu'avec le modèle indépendant. Cela prend sens dans la mesure que si les données sont générés avec un dépendance le modèle de Markov va l'interpréter lors de la restitution alors que le modèle indépent passe au travers.

Conclusion:

On remarque que l'on ne perd à priori pas notre temps à utiliser des Modèles de Markov, en effet dans le cas de signaux indepands, l'estimation de MPM via modèle de Markov n'entraine pas significativement plus d'erreur que le modèle indépendant, alors que l'inverse est fausse, l'estimation indépendante entraine ~ 2 fois plus d'erreur en moyenne que le modèle de Markov

Approfondissement:

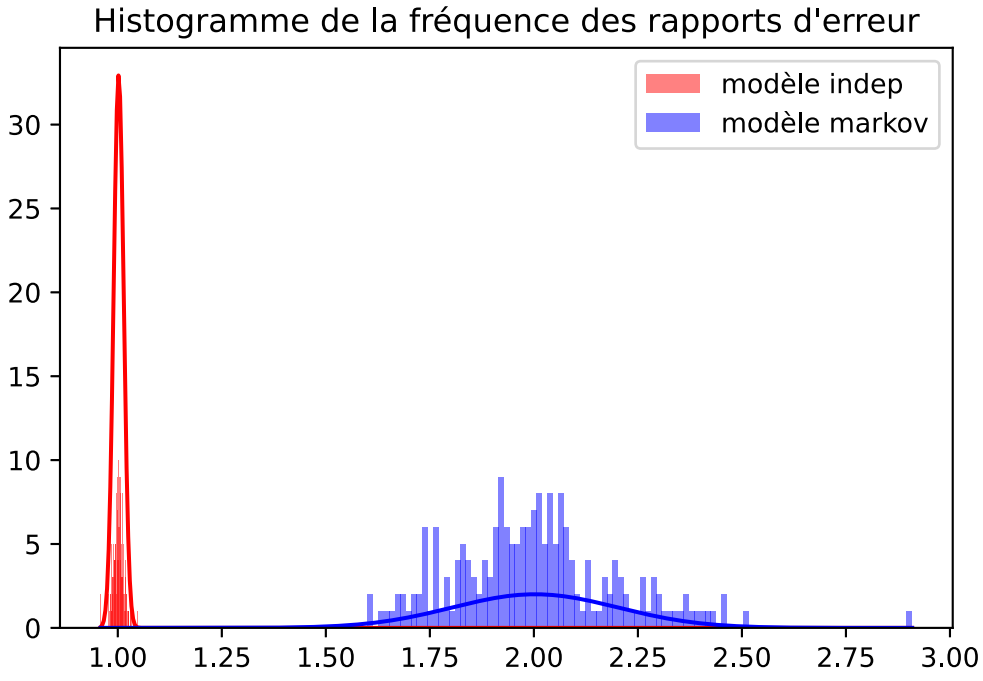
Dans les résultats précedent nous travaillons sur des obsevations qui ne sont pas consistante dans la mesure où comme on l'a fait remarquer précedament les résultats dépendent de la racine aléatoire. Je propose ainsi d'utiliser la loi des grand nombre pour justifier ma conclusion. Je realise ainsi les mêmes calculs que précedement mais pour 1000 racines aléatoire différentes. Puis je représente $\frac{erreur\ avec\ MPM\ indépendant}{erreur\ avec\ MPM\ de\ Markov}$ pour le signal modèle indépendant et modèle de Markov sous la forme d'une histogramme de fréqunece en classant le rapport d'erreur en 100 classes. On remarque alors que d'une par les distributions semblent bien être distribution gaussienne (ce qui valide ma tentative d'aller vers un LDGN), que ces gaussienne ne semblent pas se supperposer et en les "fitants" avec une gaussienne cela reste le cas, on observe qu'en sont centrés en 1 et en 2 ce qui est cohérent avec notre remarque en conclusion. Cette observation pourrait être justifié plus "mathématiquement" par un test de recouvrement de modèle mais c'est pas le sujet ici, et j'avoue ne pas savoir quel test appliquer directement.

In [7]:

```
e_indep=[]
e_markov=[]
n=1000
for seed in range(200):
    np.random.seed(seed)
    ##********************************
    ## Modèle indépendant:
    ##********************************
    ### Génération du signal:
    signal=simu_gm(n,w,p)

    ### Génération du bruit avec les différentes configurations:
    signal_noisy=[]
    for conf in config:
        signal_noisy.append( bruit_gauss(signal,w,conf[0],conf[2],conf[1],conf[3]) )

    ### Optention des paramètres d'une chaine de Markov
    p_op,A_op=calc_probaprio_mc(signal, w)

    ### Optention du nouveau signal par le mode des maginales à postériori pour le mo
dèle indep et de markov:
    new_signal_indep=[]
    new_signal_markov=[]
    i=0
    for conf in config:
        new_signal_indep.append(mpm_gm(signal_noisy[i],w,p,conf[0],conf[2],conf[1],co
nf[3]))
        new_signal_markov.append(mpm_mc(signal_noisy[i],w,p_op,A_op,conf[0],conf[2],c
onf[1],conf[3]))
        i=i+1

    ### Calcul de l'erreur:
    erreur_indep=[]
    erreur_markov=[]
    i=0
    for conf in config:
        erreur_indep.append(calc_erreur(signal,new_signal_indep[i]))
        erreur_markov.append(calc_erreur(signal,new_signal_markov[i]))
        i=i+1

    e_indep.append(np.sum(np.array(erreur_indep)/np.array(erreur_markov))/3)
    ##********************************
    ## Modèle de Markov:
    ##********************************
    ### Génération du signal:
    signal=simu_mc(n,w,p,A)

    ### Génération du bruit avec les différentes configurations:
    signal_noisy=[]
    for conf in config:
        signal_noisy.append( bruit_gauss(signal,w,conf[0],conf[2],conf[1],conf[3]) )

    ### Optention des paramètres du modèle indep
    p_indep_op=calc_probaprio_gm(signal, w)

    ### Optention du nouveau signal par le mode des maginales à postériori pour le mo
dèle indep et de markov:
    new_signal_indep=[]
    new_signal_markov=[]
    i=0
    for conf in config:
        new_signal_indep.append(mpm_gm(signal_noisy[i],w,p_indep_op,conf[0],conf[2],c
onf[1],conf[3]))
        new_signal_markov.append(mpm_mc(signal_noisy[i],w,p,A,conf[0],conf[2],conf[1
],conf[3]))
```

```
        i=i+1

    ### Calcul de l'erreur:
    erreur_indep=[]
    erreur_markov=[]
    i=0
    for conf in config:
        erreur_indep.append(calc_erreur(signal,new_signal_indep[i]))
        erreur_markov.append(calc_erreur(signal,new_signal_markov[i]))
        i=i+1
    e_markov.append(np.sum(np.array(erreur_indep)/np.array(erreur_markov))/3)

e_indep=np.array(e_indep)
e_markov=np.array(e_markov)
```

In [8]:
```
from scipy.stats import norm
mu_m,sig_m=norm.fit(e_markov)
mu_i,sig_i=norm.fit(e_indep)

min=np.min(np.concatenate((e_markov,e_indep)))
max=np.max(np.concatenate((e_markov,e_indep)))
x = np.linspace(min,max, 500)
p_m = norm.pdf(x, mu_m, sig_m)
p_i = norm.pdf(x, mu_i, sig_i)

plt.hist(e_indep,bins=100,color="red",alpha=0.5,label="modèle indep")
plt.plot(x,p_i,color="red")
plt.hist(e_markov,bins=100,color="blue",alpha=0.5,label="modèle markov")
plt.plot(x,p_m,color="blue")
plt.title("Histogramme de la fréquence des rapports d'erreur")
plt.legend()
plt.show()

print(mu_i)
print(mu_m)
```



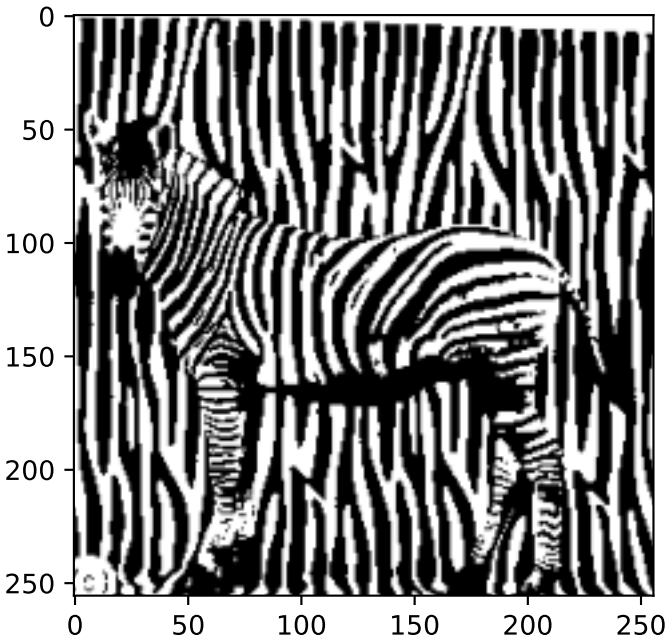

```
1.0020815482647434
2.002811040493563
```

3. Application à la segmentation d'images

On applique maintenant l'estimation à une image 2D. L'objectif etant de retrouver à partir d'une image buitée, l'image d'origine mais *sans connaitre les paramètre des modèles* contrairement à ce que nous avons fait précedement. Afin de restituer les paramètre nous allons utilisaer la methode EM (expectation maximisation). Pour illustrer le code, nous allons traiter l'image de zebre. (Puis toute les autres à la fin). Nous chargons cette image avec la transformation de Peano qui empiriquement conserve le maximum d'informations locales de l'image.

Importation de l'image

In [9]:
```
import numpy as np
import cv2 as cv
from tools import bruit_gauss,calc_erreur, peano_transform_img, transform_peano_in_im
g, line_transform_img, transform_line_in_img
from gaussian_mixture import *
from markov_chain import *
from matplotlib import pyplot as plt

img=cv.imread("../images/zebre2.bmp",cv.IMREAD_GRAYSCALE )
plt.imshow(img, cmap='gray')
plt.show()
print(img.shape)
print(img.size)
signal_image=peano_transform_img(img)
print(signal_image.shape)
```

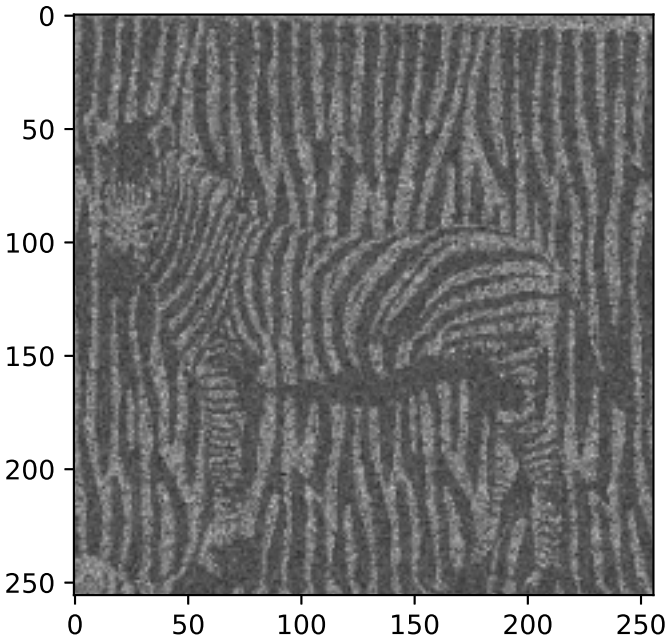



```
(256, 256)
65536
(65536,)
```

Bruitage de l'image par vecteur gaussien

Bruitons l'image obtenue avec une bruit gaussiens qui sera inconue dans la suite du raisonnement.

In [10]:
```
m1=0
sig1=1
m2=3
sig2=2
w=np.array([0,255])
signal_noisy_image=bruit_gauss(signal_image,w,m1,sig1,m2,sig2)
img_noisy=transform_peano_in_img(signal_noisy_image,256)
plt.imshow(img_noisy, cmap='gray')
plt.show()
```



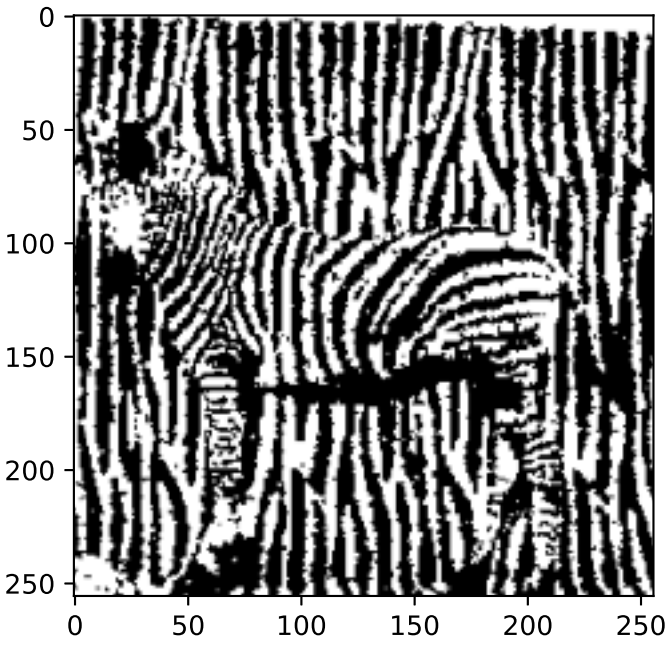
Restoration de l'image à l'aide d'un MPM suivant la chane de Markov

A partir de ce signal bruité, l'objectif est d'estimer les paramètres de la loi de la chaine de Markov et du bruit, afin de restaurer le signal.

In [11]:
```
(p_est, A_est, m1_est, sig1_est, m2_est, sig2_est)=estim_param_EM_mc(10,signal_noisy_image,p,A,m1,sig1,m2,sig2)
new_signal_image=mpm_mc(signal_noisy_image,w,p_est, A_est, m1_est, sig1_est, m2_est, sig2_est)
```

```
itération 0 sur 10
{'    p': array([0.53857763, 0.46142237]), 'A': array([[0.89352764, 0.10647236],
       [0.12427996, 0.87572004]]), 'm1': -0.01937975048248041, 'sig1': 0.9869300473947437, 'm2': 2.844776608546165, 'sig2': 2.0634490470139077}
itération 1 sur 10
{'    p': array([0.5419082, 0.4580918]), 'A': array([[0.89704985, 0.10295015],
       [0.12179093, 0.87820907]]), 'm1': -0.007636091941293445, 'sig1': 0.9896269386155722, 'm2': 2.851708117496625, 'sig2': 2.0724663469668787}
itération 2 sur 10
{'    p': array([0.54257446, 0.45742554]), 'A': array([[0.89893451, 0.10106549],
       [0.11988273, 0.88011727]]), 'm1': -0.003475362541239346, 'sig1': 0.9922779068484447, 'm2': 2.8509376291758812, 'sig2': 2.0760558346316675}
itération 3 sur 10
{'    p': array([0.54252579, 0.45747421]), 'A': array([[0.90005692, 0.09994308],
       [0.1185281 , 0.8814719 ]]), 'm1': -0.0016578573585655247, 'sig1': 0.9937189752889752, 'm2': 2.848478559784023, 'sig2': 2.0783218426047965}
itération 4 sur 10
{'    p': array([0.54226009, 0.45773991]), 'A': array([[0.90075625, 0.09924375],
       [0.1175728 , 0.8824272 ]]), 'm1': -0.0007961956211054281, 'sig1': 0.9944109775994586, 'm2': 2.84580342247709, 'sig2': 2.0799960589216973}
itération 5 sur 10
{'    p': array([0.54195211, 0.45804789]), 'A': array([[0.90120688, 0.09879312],
       [0.11689383, 0.88310617]]), 'm1': -0.000378732654740904, 'sig1': 0.9947182243374515, 'm2': 2.8433955219088434, 'sig2': 2.081266633849556}
itération 6 sur 10
{'    p': array([0.54166846, 0.45833154]), 'A': array([[0.90150605, 0.09849395],
       [0.11640676, 0.88359324]]), 'm1': -0.0001772293184749538, 'sig1': 0.9948394745804856, 'm2': 2.8413974280412453, 'sig2': 2.082227795575304}
itération 7 sur 10
{'    p': array([0.54143062, 0.45856938]), 'A': array([[0.90170984, 0.09829016],
       [0.11605467, 0.88394533]]), 'm1': -8.178363466776505e-05, 'sig1': 0.9948750686094201, 'm2': 2.8398109025778218, 'sig2': 2.082949700407002}
itération 8 sur 10
{'    p': array([0.54124072, 0.45875928]), 'A': array([[0.90185159, 0.09814841],
       [0.11579869, 0.88420131]]), 'm1': -3.816450794108886e-05, 'sig1': 0.9948739748223181, 'm2': 2.8385838924556595, 'sig2': 2.083488657890654}
itération 9 sur 10
{'    p': array([0.54109337, 0.45890663]), 'A': array([[0.90195179, 0.09804821],
       [0.11561185, 0.88438815]]), 'm1': -1.950262292346463e-05, 'sig1': 0.9948593251516704, 'm2': 2.8376504822893165, 'sig2': 2.0838892916249594}
```

In [12]:
```
img_restaure=transform_peano_in_img(new_signal_image,256)
plt.imshow(img_restaure, cmap='gray')
plt.show()
erreur=calc_erreur(signal_image,new_signal_image)
print("Erreur commise: {:.2%}".format(erreur))
```



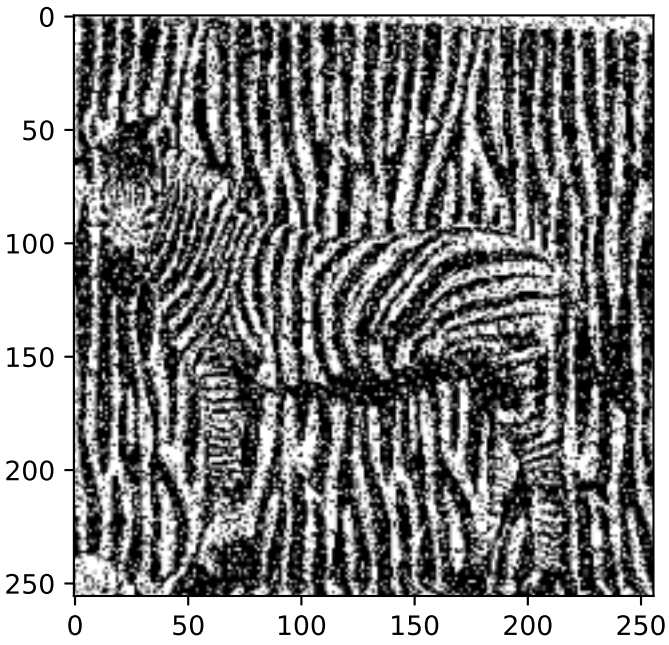

```
Erreur commise: 8.31%
```

Reconstitution de l'image par MPM suivant le modèle indépendant

In [13]:
```
(p_est, m1_est, sig1_est, m2_est, sig2_est)=estim_param_EM_gm(10,signal_noisy_image,p
,m1,sig1,m2,sig2)
new_signal_image=mpm_gm(signal_noisy_image,w,p_est, m1_est, sig1_est, m2_est, sig2_es
t)
```

```
{'p': array([0.39349065, 0.60650935]), 'm1': -0.1697435558193064, 'sig1': 0.91161051
76453916, 'm2': 2.2571763236962665, 'sig2': 2.1467856581954785}
{'p': array([0.41138624, 0.58861376]), 'm1': -0.1366225600889087, 'sig1': 0.87675912
97898911, 'm2': 2.307813367762246, 'sig2': 2.167269735459815}
{'p': array([0.42426006, 0.57573994]), 'm1': -0.11944626899578599, 'sig1': 0.8666141
308334526, 'm2': 2.3498149974436244, 'sig2': 2.1735267998157064}
{'p': array([0.43411958, 0.56588042]), 'm1': -0.10913286221484204, 'sig1': 0.8668309
617805583, 'm2': 2.3849257056826314, 'sig2': 2.1736332820302513}
{'p': array([0.44224656, 0.55775344]), 'm1': -0.10205568639277529, 'sig1': 0.8712872
537122878, 'm2': 2.415654887635327, 'sig2': 2.1708997429510255}
{'p': array([0.44932157, 0.55067843]), 'm1': -0.09661155164658076, 'sig1': 0.8772816
337496545, 'm2': 2.4435598627089328, 'sig2': 2.1668349400791027}
{'p': array([0.45568705, 0.54431295]), 'm1': -0.09206068941327256, 'sig1': 0.8836346
633517848, 'm2': 2.469456080552369, 'sig2': 2.162157358246665}
{'p': array([0.46151791, 0.53848209]), 'm1': -0.08805332842906745, 'sig1': 0.8898521
548758827, 'm2': 2.4937584000191686, 'sig2': 2.1572165247339212}
{'p': array([0.46691006, 0.53308994]), 'm1': -0.08441890704106933, 'sig1': 0.8957451
538703647, 'm2': 2.5166899333915445, 'sig2': 2.1521859270401915}
{'p': array([0.47192257, 0.52807743]), 'm1': -0.0810694821457361, 'sig1': 0.90125735
35926767, 'm2': 2.538386424168824, 'sig2': 2.147154466055066}
```

In [14]:
```
img_restaure=transform_peano_in_img(new_signal_image,256)
plt.imshow(img_restaure, cmap='gray')
plt.show()
erreur=calc_erreur(signal_image,new_signal_image)
print("Erreur commise: {:.2%}".format(erreur))
```




```
Erreur commise: 14.91%
```

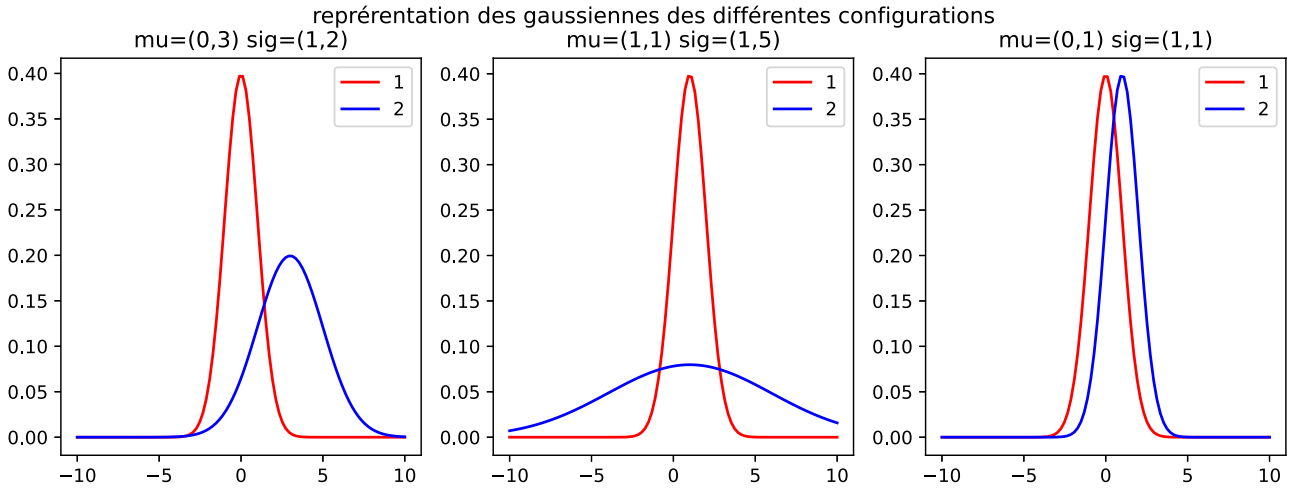
Nous remarquons que comme nous constaté au debut de cette partie, nous avons une erreur deux fois plus importante sur le modèle indépendant. Les images ne semblent pas dérroger à cette règle.

Resultat suivant les différentes configurations

Etudions maintenant les résultats pour les différantes configuration et images, pour eviter la surcharge de code ici, le fichier "ttm_image.py" contient le code necessaire pour efectuer le traitement précedement décrit aux images.

On remarque tout d'abord que sur toutes les images qui sont traités la première est celle qui génère le moins d'erreur et la troisième celle qui en génère le plus. Les configurations correpondent à des configurations différantes de bruit, ainsi la restitution se fait mieux pour des bruits de sigma faibles avec des moyennes différentes qu'une configuration des moyennes et des sigmas très proches. Ce resultat n'a rien d'abérent à mes yeux par rapport à ce que l'on pouvait s'attendre au regard des distributions:

In [15]:
```
from scipy.stats import norm
plt.figure(figsize=(12, 4))
x = np.linspace(-10,10,100)
for i in range(3):
    conf=config[i]
    norm1 = norm.pdf(x, conf[0], conf[2])
    norm2 = norm.pdf(x, conf[1], conf[3])
    plt.subplot(131+i)
    plt.plot(x,norm1,color="red",label="1")
    plt.plot(x,norm2,color="blue",label="2")
    plt.title('mu=({},{}) sig=({},{})'.format(config[i][0],config[i][1],config[i][2],
config[i][3]))
    plt.legend()
plt.suptitle("reprérentation des gaussiennes des différentes configurations")
plt.show()
```



On marque au fil des configurations les distributions sont de plus en plus supperposés et donc de moins en moins dissernable.

De manière globale on remarque que les images avec de grandes surface de même couleur sont mieux restitué que des images avec beaucoup de variations. Par exemple, le B, la cible, les lettres ou le veau sont dans ces configurations de grandes surfaces monochrome et ont des erreurs faible de l'ordre de 2%, tandis que promenade, country, city ou zèbre ont beaucoup plus de variations et on des erreurs 8 à 10% en moyenne. Cela s'explique à mes yeux par le fait que plus la variation spatiale est élevé plus la fréquence des changements d'un pixel à l'autre est élevée. Cela peut amener au même qu'une reconstruction indépendante et met en défaut l'hypothèse de stationarité. En annexe 1 j'ai fait la dérivée de B et Zebre (peu/beaucoup de variation visuelle) on remarque ainsi que le nombre de pixel blanc de la dérivé de zebre est bien plus élevé que B ce qui représente mathématiquement ce que j'appelle la variation. On remarque egalement en annexe 2 que les coefficient de A sont beaucoup séparés pour Zebre que pour B ce qui peu corroborrer au fait que l'on se rapproche un peu plus du cas indépendant pour zebre que pour B par exemple.

Pour ce qui est des résultats du modèle indépendant on retrouve dans le cas d'images "avec de fortes variation" le résultat avec une erreur doublée par rapport au modèle de Markov. Par contre on remarque que sur des images avec peu de variation les résultats du modèle indépendant sont catastrophiques, parfois 10 fois moins bon. C'est à mes yeux cohérent car dans le cas de ses images la probabilité de transition d'un etat chromatique à un autre (différent) est relativement élevé et Markov apporte beaucoup d'avantage alors que le modèle indépendant ne fait pas la différence.

Le zèbre:

In [16]:

```
from ttm_image import ttm
ttm("zebre2.bmp")
```

Image d'origine:

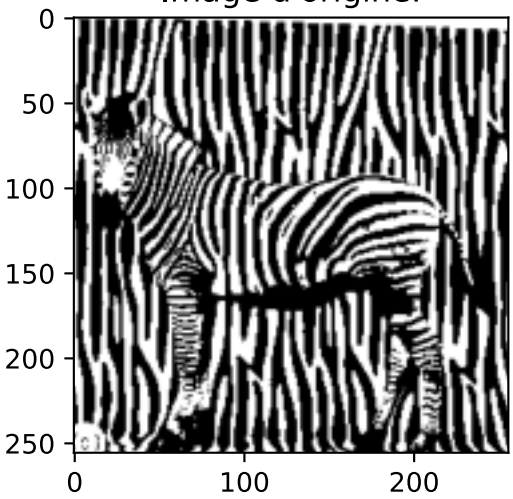
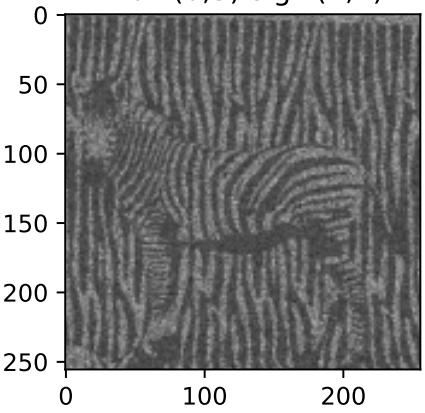
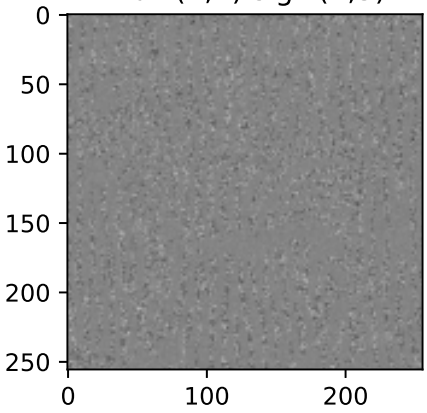


Image bruité, en fonction de la configuration du bruit:

mu=(0,3) sig=(1,2)



mu=(1,1) sig=(1,5)



mu=(0,1) sig=(1,1)

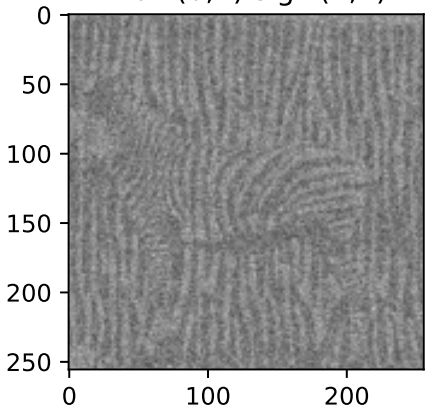
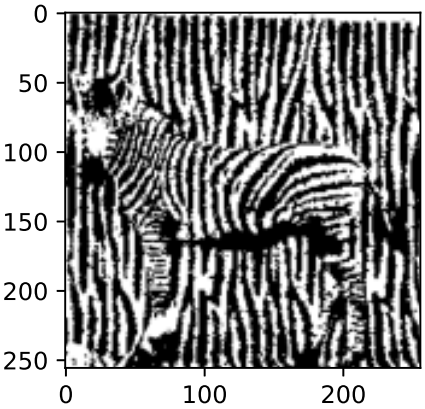
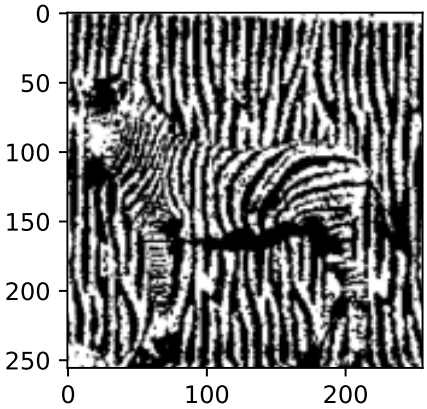


Image restitué, en fonction de la configuration du bruit, pour le modèle de Markov:

Erreur:8.30%



Erreur:9.69%



Erreur:20.20%

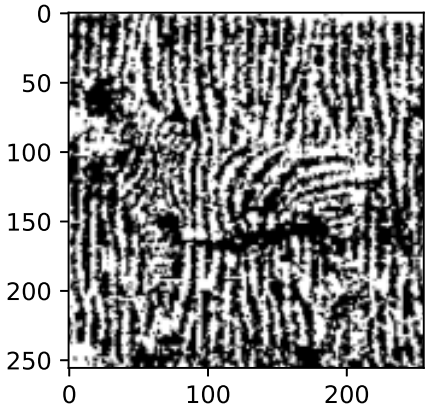
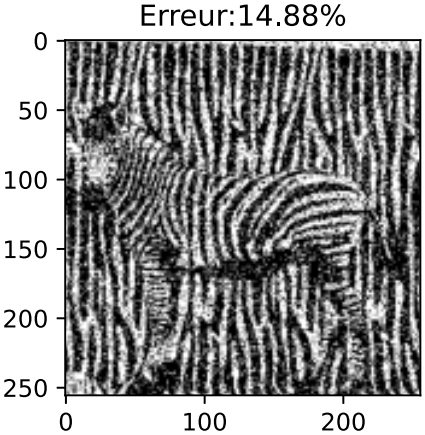
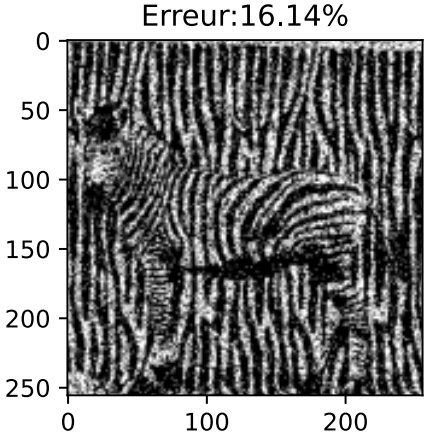


Image restitué, en fonction de la configuration du bruit, pour le modèle indépendant:

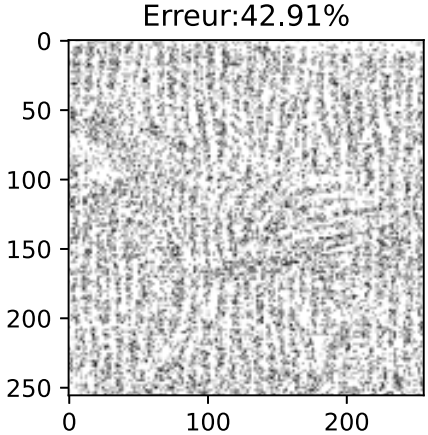
Erreur:14.88%



Erreur:16.14%



Erreur:42.91%



Lettres;

In [17]:
```
ttm("alfa2.bmp")
```

Image d'origine:

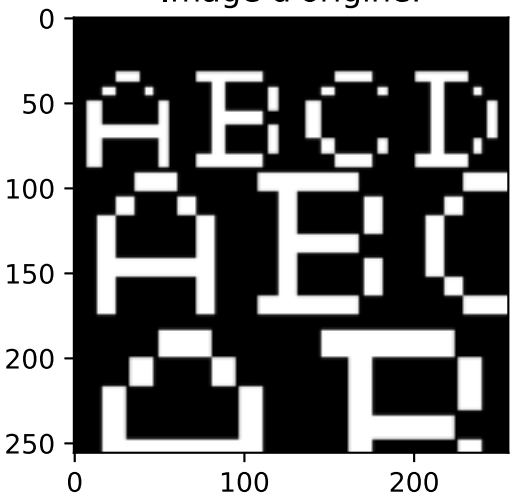
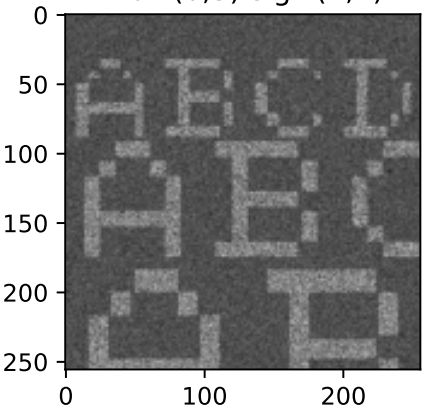
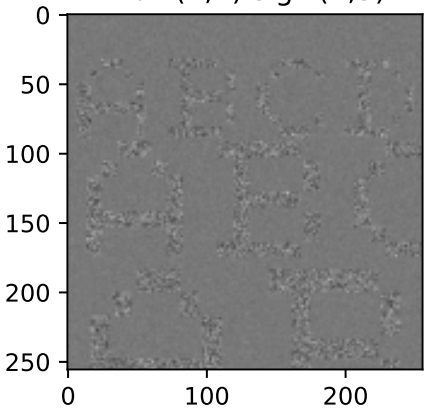


Image bruité, en fonction de la configuration du bruit:

mu=(0,3) sig=(1,2)



mu=(1,1) sig=(1,5)



mu=(0,1) sig=(1,1)

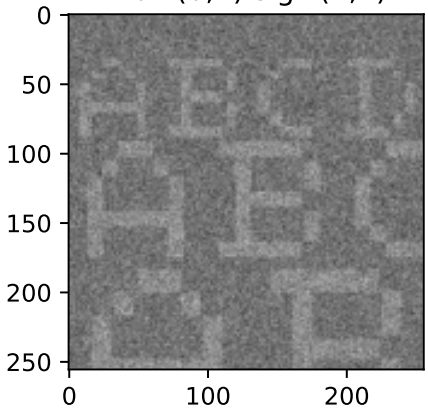
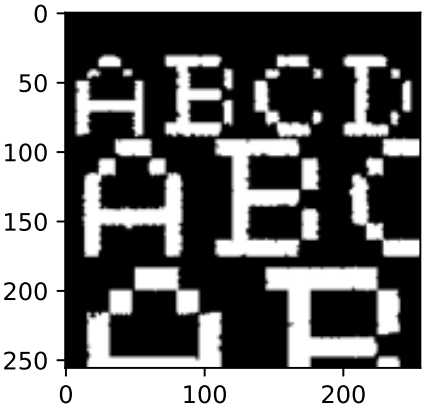
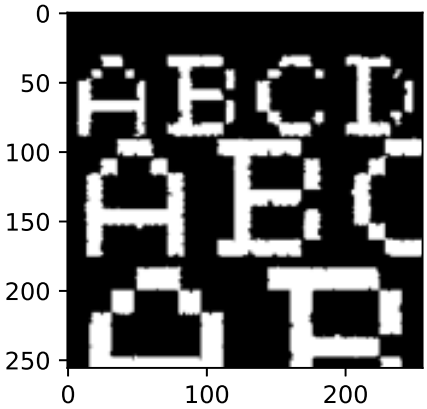


Image restitué, en fonction de la configuration du bruit, pour le modèle de Markov:

Erreur:2.59%



Erreur:2.72%



Erreur:5.18%

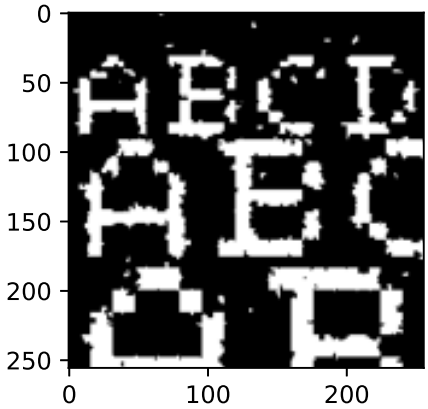
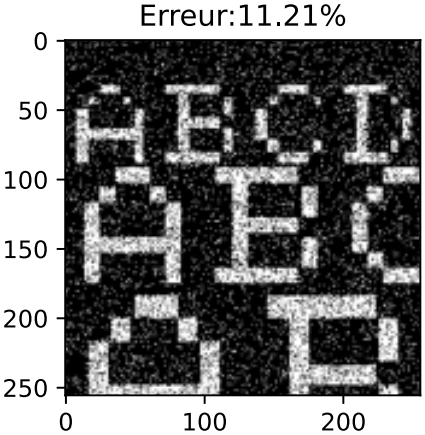
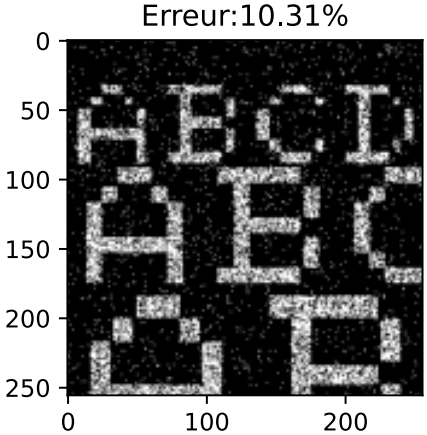


Image restitué, en fonction de la configuration du bruit, pour le modèle indépendant:

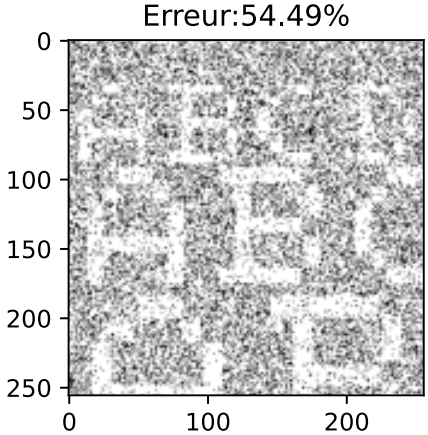
Erreur:11.21%



Erreur:10.31%



Erreur:54.49%



Lettre B:

In [18]:
```
ttm("beee2.bmp")
```

Image d'origine:

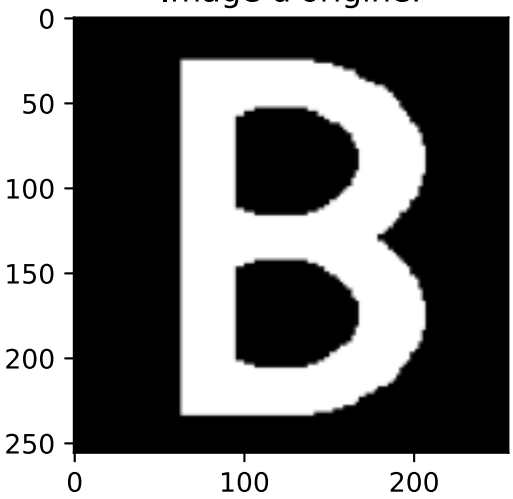
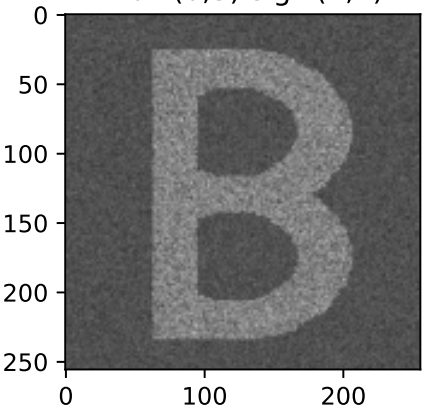
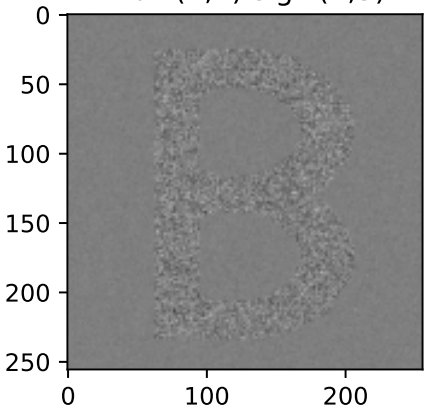


Image bruité, en fonction de la configuration du bruit:

mu=(0,3) sig=(1,2)



mu=(1,1) sig=(1,5)



mu=(0,1) sig=(1,1)

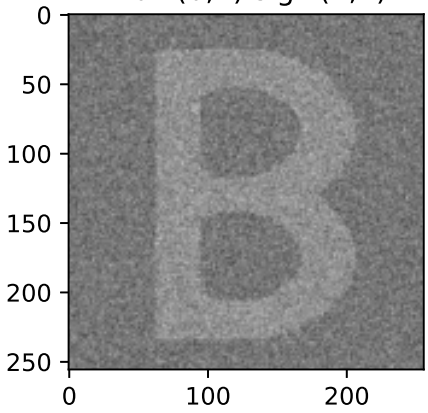
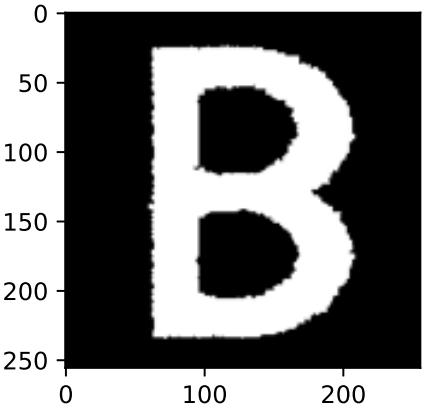
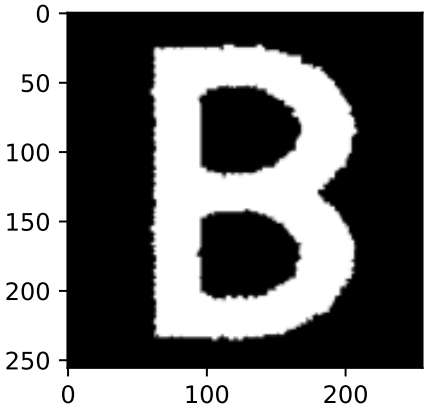


Image restitué, en fonction de la configuration du bruit, pour le modèle de Markov:

Erreur:1.01%



Erreur:1.16%



Erreur:2.16%

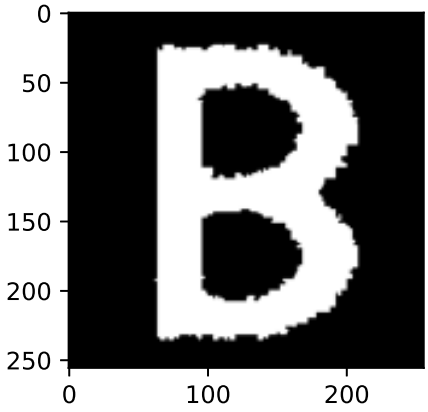
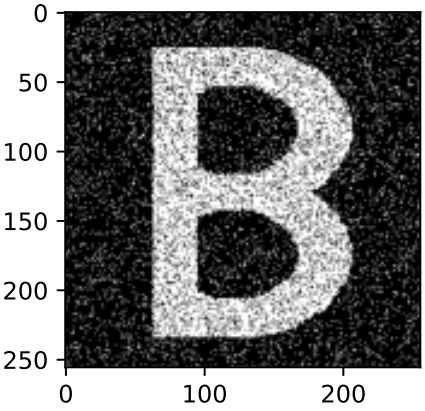
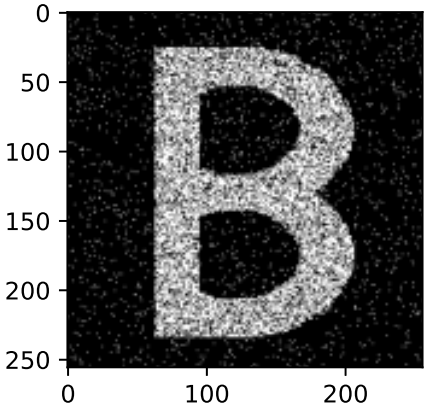


Image restitué, en fonction de la configuration du bruit, pour le modèle indépendant:

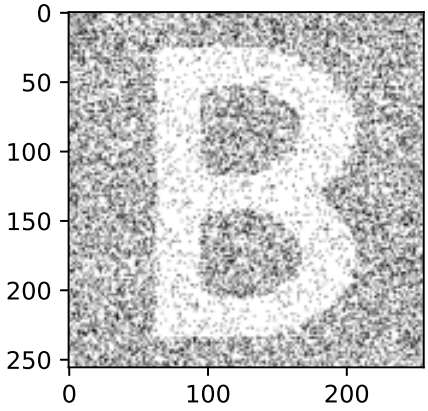
Erreur:12.48%



Erreur:11.83%



Erreur:51.74%



Cible

In [19]:
```
ttm("cible2.bmp")
```

Image d'origine:

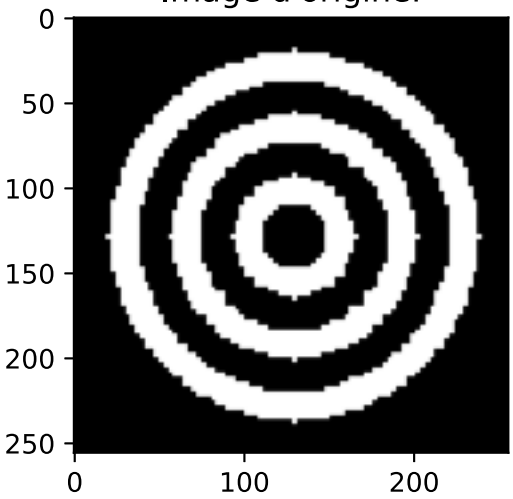
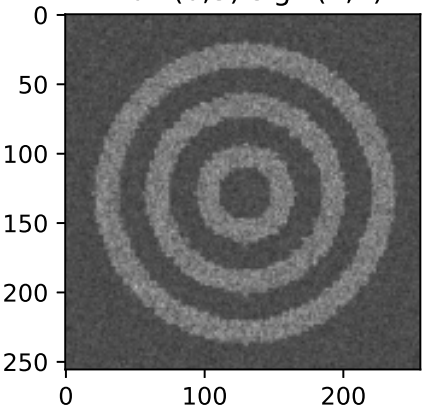
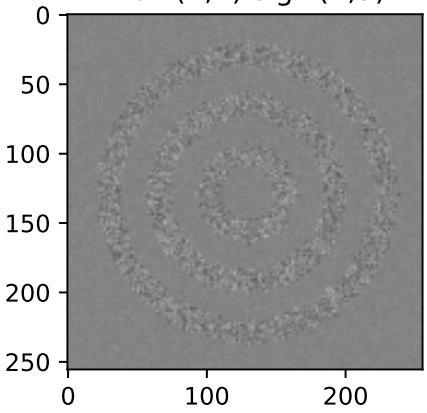


Image bruité, en fonction de la configuration du bruit:

mu=(0,3) sig=(1,2)



mu=(1,1) sig=(1,5)



mu=(0,1) sig=(1,1)

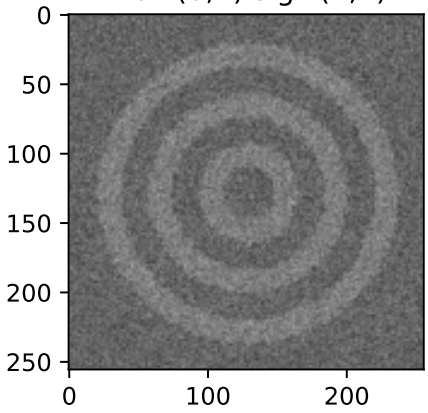
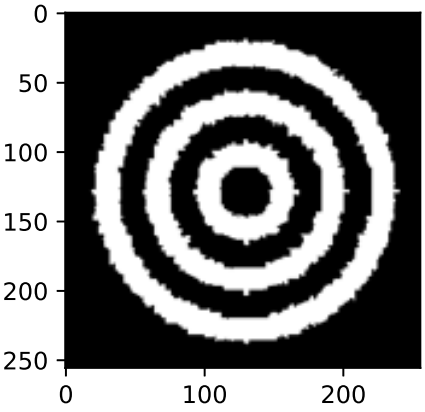
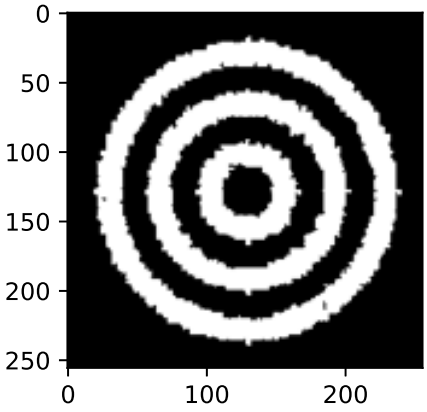


Image restitué, en fonction de la configuration du bruit, pour le modèle de Markov:

Erreur:2.08%



Erreur:2.27%



Erreur:4.07%

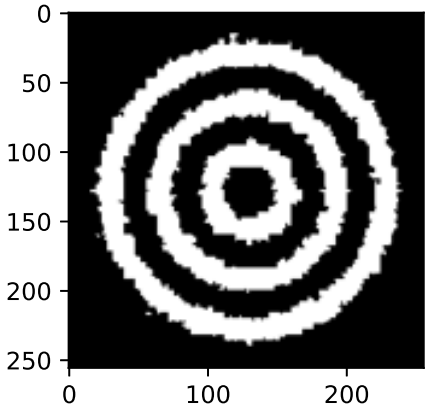
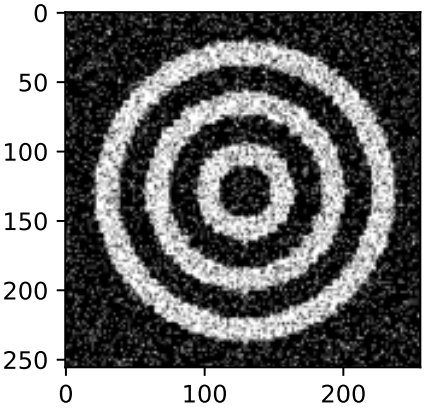
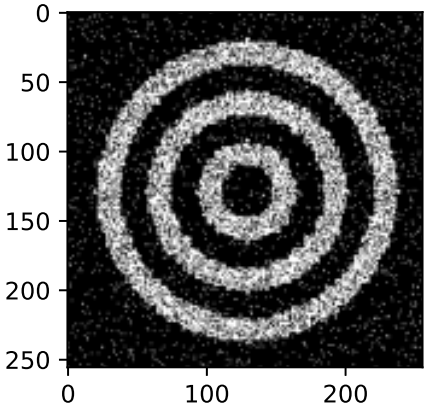


Image restitué, en fonction de la configuration du bruit, pour le modèle indépendant:

Erreur:12.70%



Erreur:12.44%



Erreur:50.91%

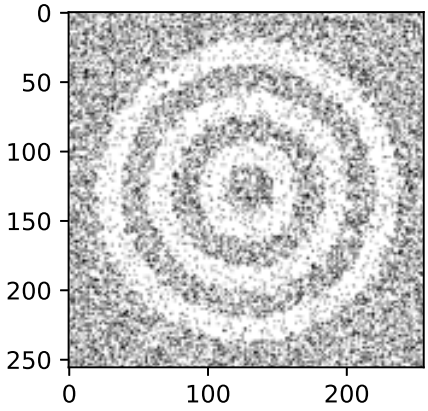


Image satellite de ville

In [20]:
```
ttm("city2.bmp")
```

Image d'origine:

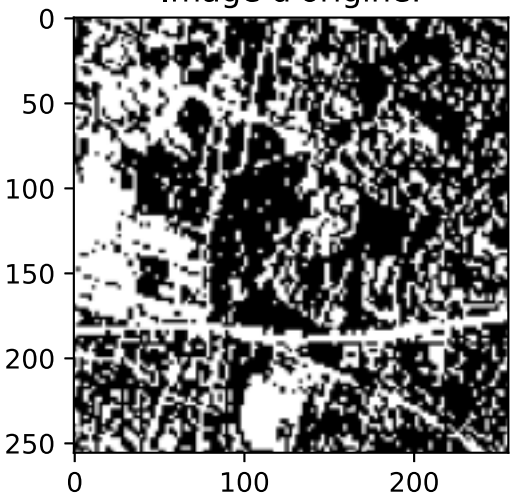
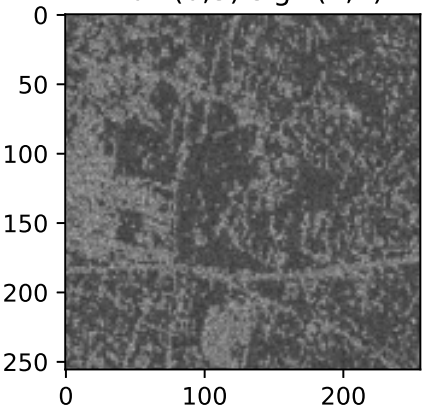
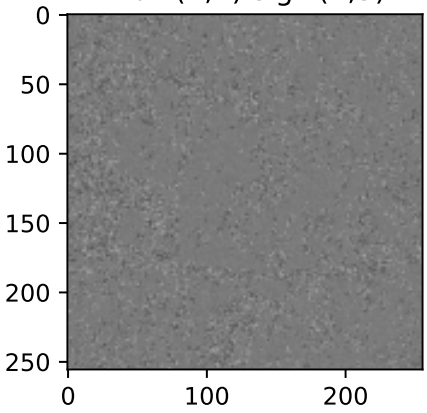


Image bruité, en fonction de la configuration du bruit:

mu=(0,3) sig=(1,2)



mu=(1,1) sig=(1,5)



mu=(0,1) sig=(1,1)

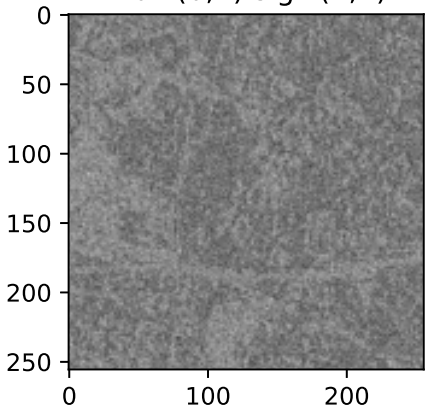
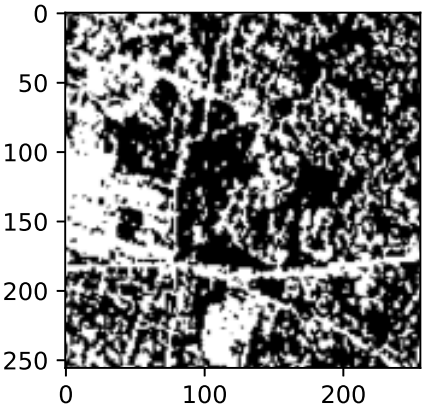
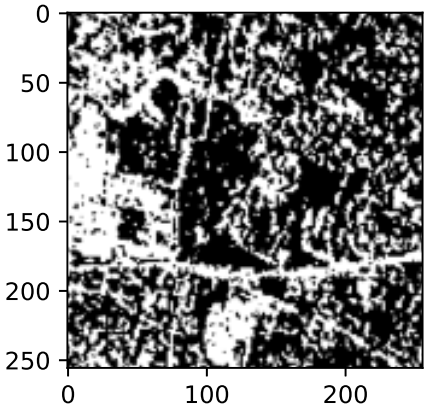


Image restitué, en fonction de la configuration du bruit, pour le modèle de Markov:

Erreur:8.51%



Erreur:9.57%



Erreur:19.08%

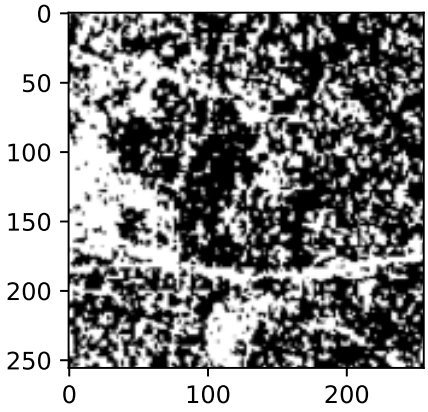
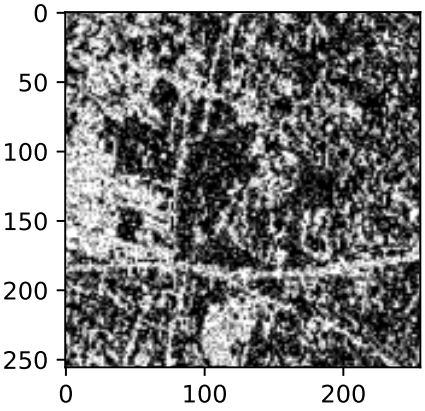
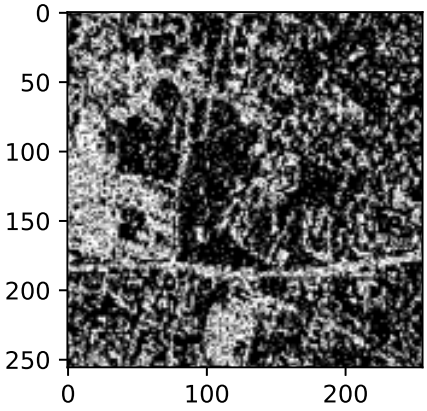


Image restitué, en fonction de la configuration du bruit, pour le modèle indépendant:

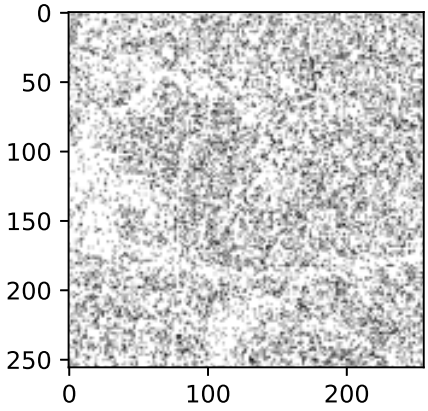
Erreur:14.74%



Erreur:15.48%



Erreur:44.75%



In [21]:
```
## "Country"
```

In [22]:
```
ttm("country2.bmp")
```

Image d'origine:

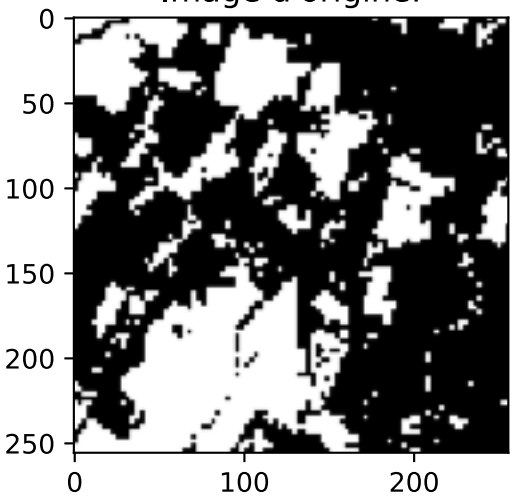


Image bruité, en fonction de la configuration du bruit:

mu=(0,3) sig=(1,2) mu=(1,1) sig=(1,5) mu=(0,1) sig=(1,1)

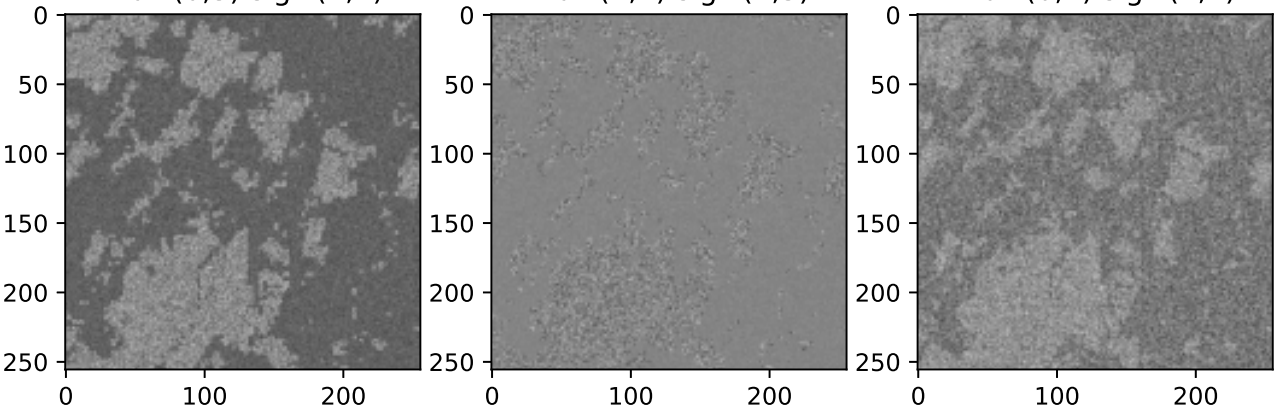


Image restitué, en fonction de la configuration du bruit, pour le modèle de Markov:

Erreur:3.50% Erreur:4.15% Erreur:8.01%

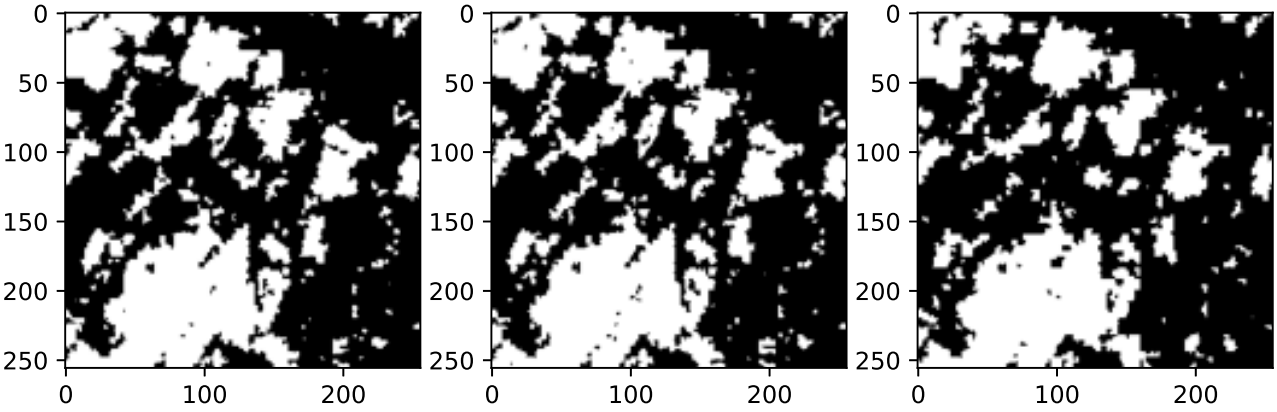
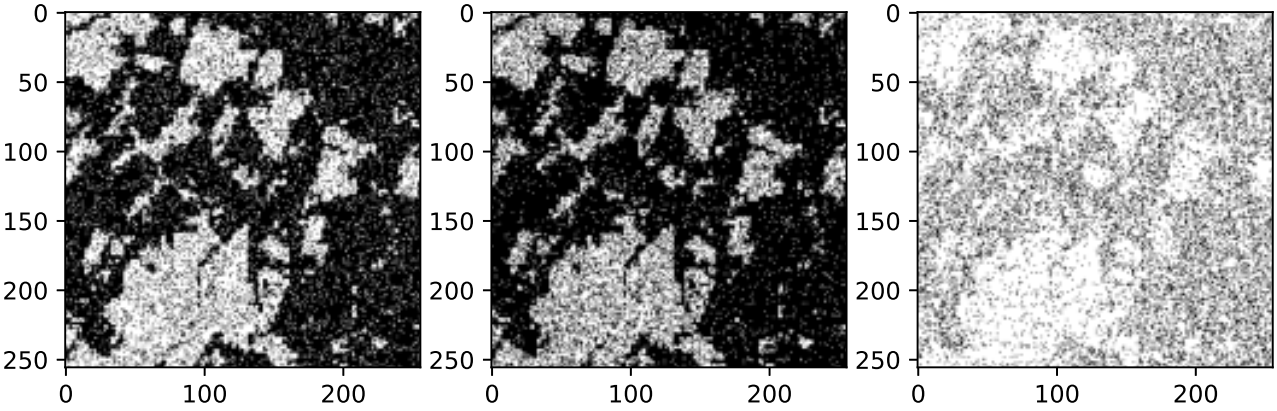


Image restitué, en fonction de la configuration du bruit, pour le modèle indépendant:

Erreur:13.13% Erreur:13.95% Erreur:48.31%



Promenade

In [23]:
```
ttm("promenade2.bmp")
```

Image d'origine:

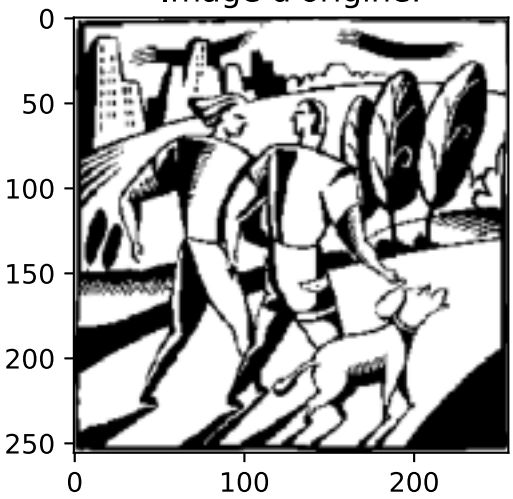


Image bruité, en fonction de la configuration du bruit:

mu=(0,3) sig=(1,2) mu=(1,1) sig=(1,5) mu=(0,1) sig=(1,1)

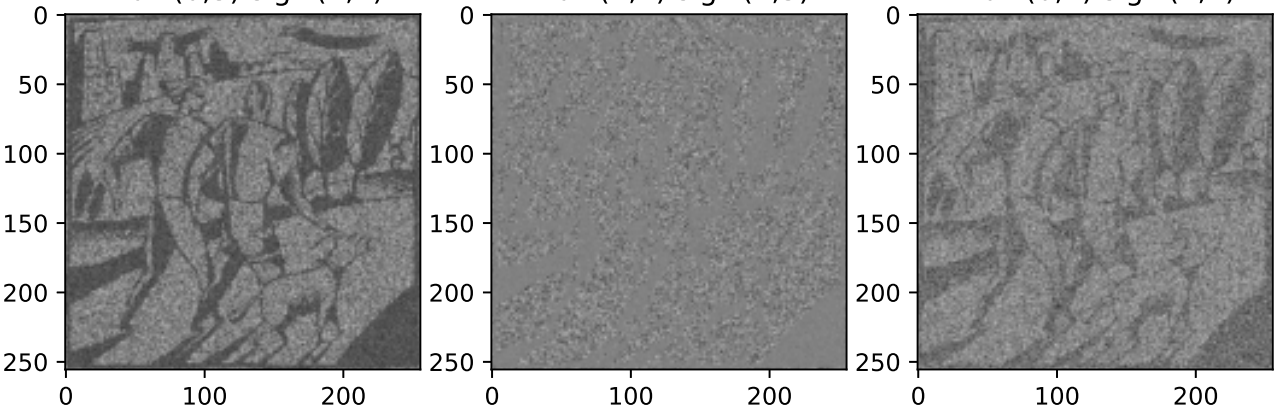


Image restitué, en fonction de la configuration du bruit, pour le modèle de Markov:

Erreur:7.69% Erreur:8.69% Erreur:14.51%

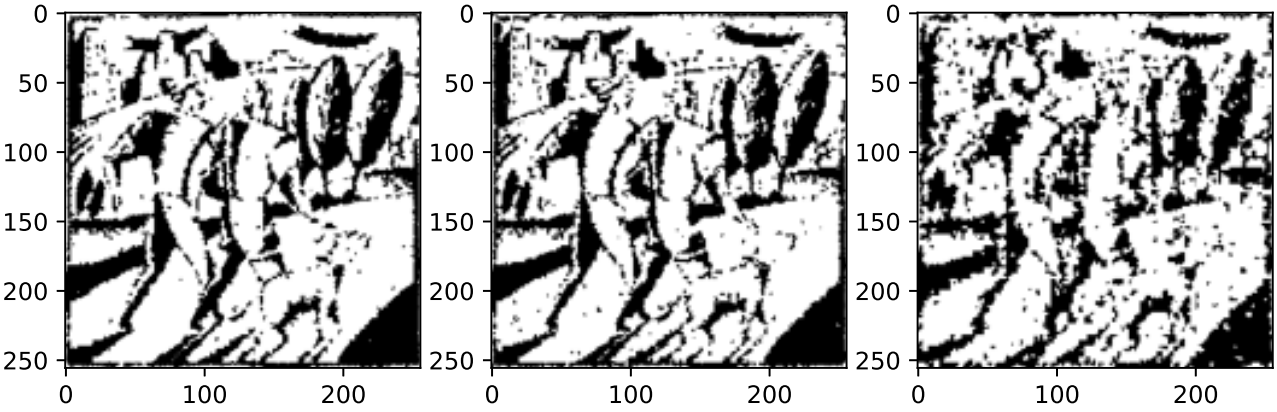
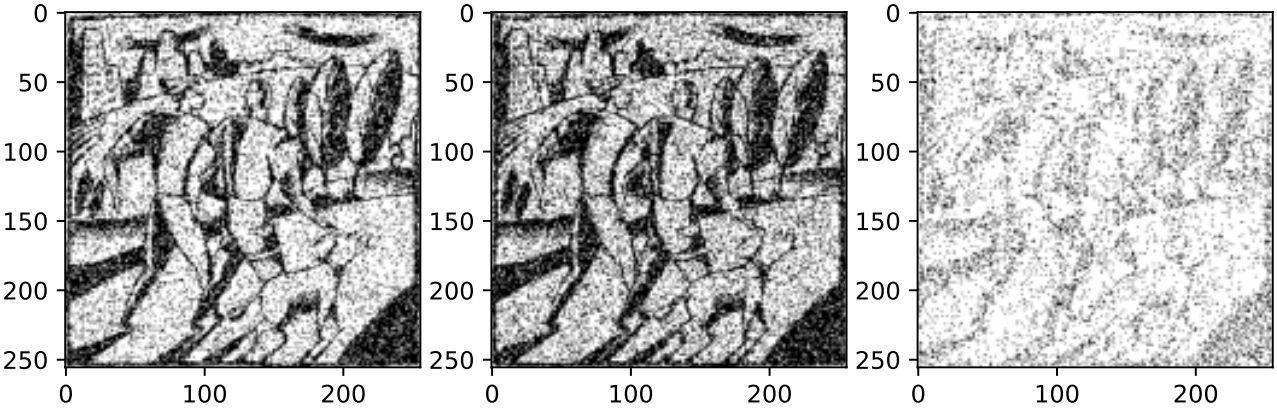


Image restitué, en fonction de la configuration du bruit, pour le modèle indépendant:

Erreur:16.15% Erreur:19.83% Erreur:30.65%



Veau

In [24]:
```
ttm("veau2.bmp")
```

Image d'origine:

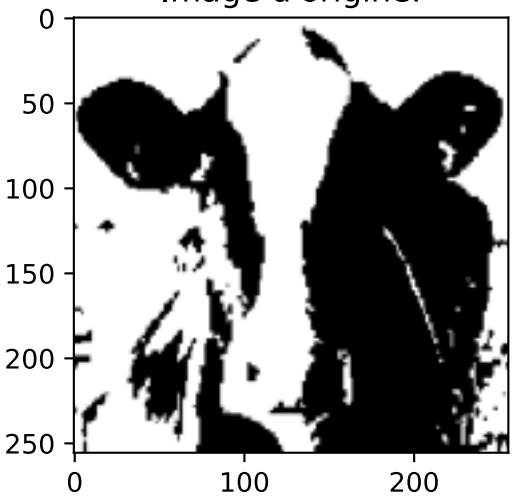


Image bruité, en fonction de la configuration du bruit:

mu=(0,3) sig=(1,2) mu=(1,1) sig=(1,5) mu=(0,1) sig=(1,1)

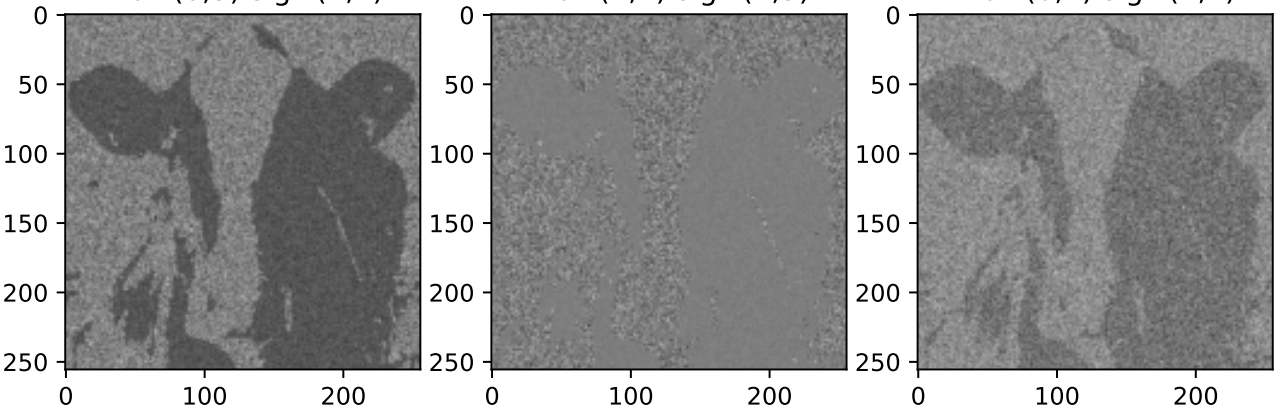


Image restitué, en fonction de la configuration du bruit, pour le modèle de Markov:

Erreur:2.43% Erreur:2.73% Erreur:5.26%

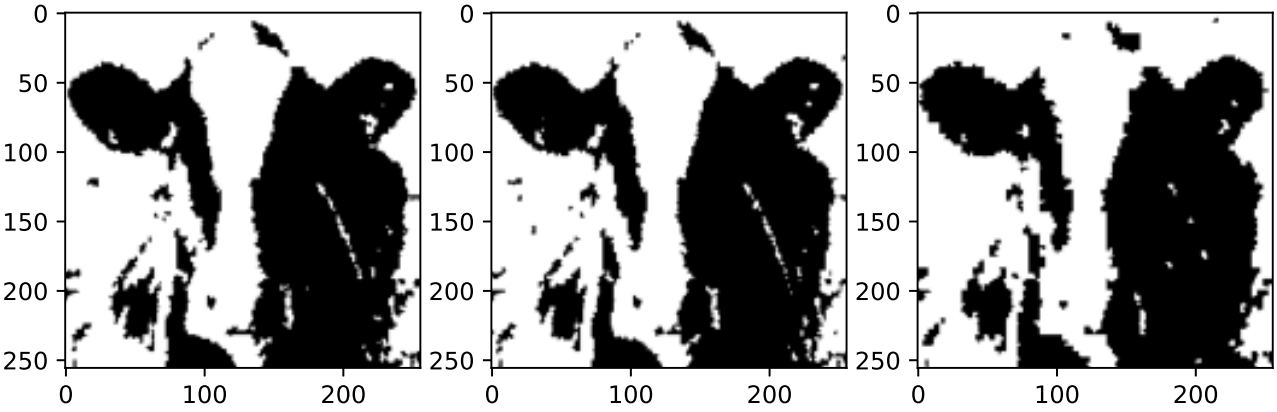
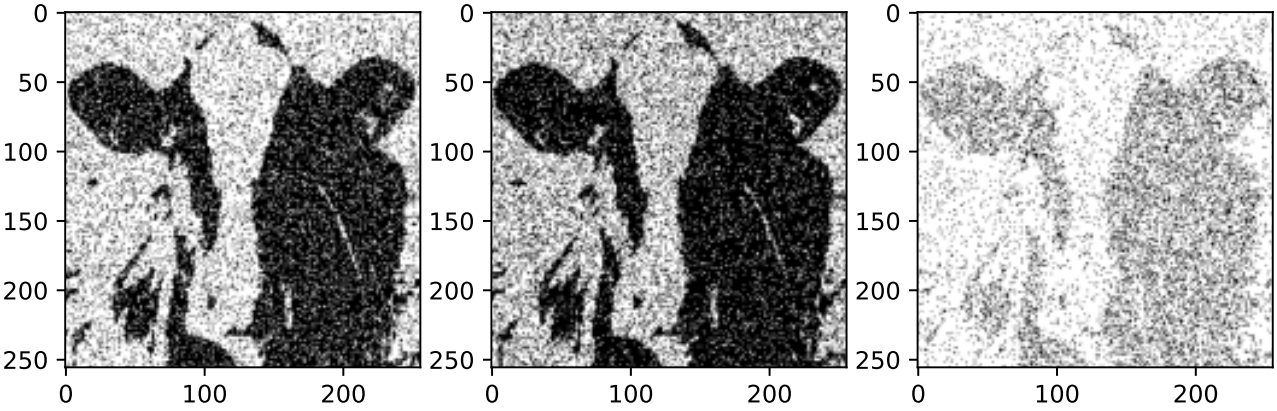


Image restitué, en fonction de la configuration du bruit, pour le modèle indépendant:

Erreur:15.84% Erreur:18.37% Erreur:37.06%

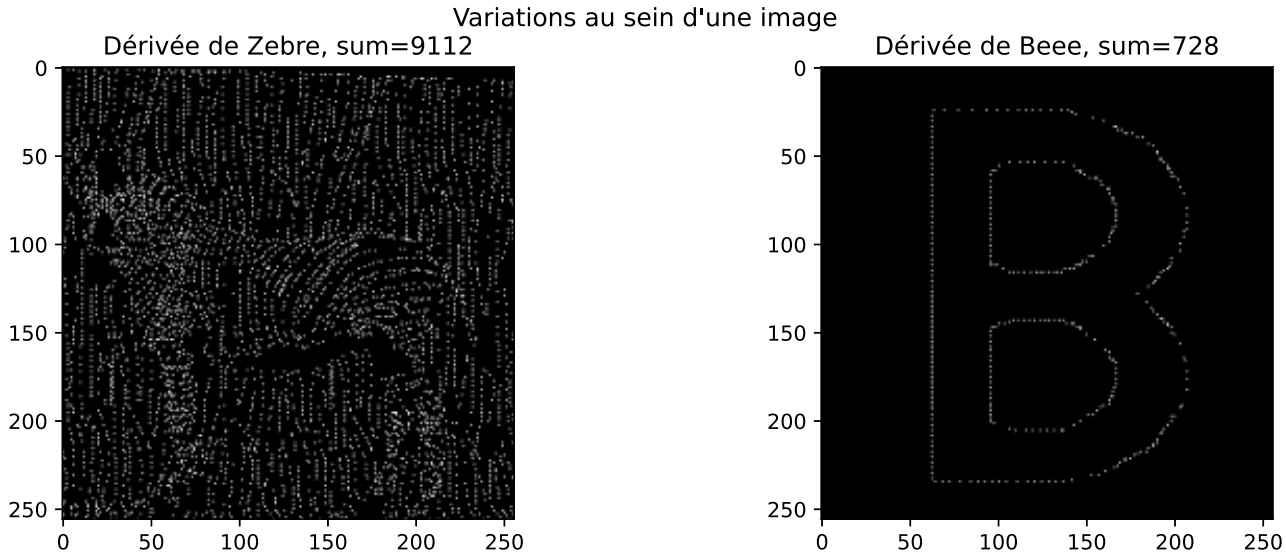


Approfondissement:

Annexe 1: dérivation de l'image dans le sens de Peano

In [25]:
```
from markov_chain import *
import cv2 as cv
from tools import bruit_gauss,calc_erreur, peano_transform_img, transform_peano_in_im
g, line_transform_img, transform_line_in_img
from ttm_image import *

img_z=cv.imread("../images/zebre2.bmp",cv.IMREAD_GRAYSCALE )
signal_z=peano_transform_img(img_z)
img_b=cv.imread("../images/beee2.bmp",cv.IMREAD_GRAYSCALE )
signal_b=peano_transform_img(img_b)

Z=[signal_z[0]]
B=[signal_b[0]]
sz=0
sb=0
for k in range(1,len(signal_b)):
    Z.append(np.abs(signal_z[k-1]-signal_z[k]))
    sz+=(Z[k]!=0)*1
    B.append(np.abs(signal_b[k-1]-signal_b[k]))
    sb+=(B[k]!=0)*1
Z=np.array(Z)
B=np.array(B)
Z=transform_peano_in_img(Z,256)
B=transform_peano_in_img(B,256)

plt.figure(figsize=(12, 4))
plt.subplot(121)
plt.imshow(Z, cmap='gray')
plt.title("Dérivée de Zebre, sum={}".format(sz))
plt.subplot(122)
plt.imshow(B, cmap='gray')
plt.title("Dérivée de Beee, sum={}".format(sb))
plt.suptitle("Variations au sein d'une image")
```

Out[25]:
```
Text(0.5, 0.98, "Variations au sein d'une image")
```



Annexe 2: impact de la variation des pixels sur A

In [26]:
```
from markov_chain import *
import cv2 as cv
from tools import bruit_gauss,calc_erreur, peano_transform_img, transform_peano_in_im
g, line_transform_img, transform_line_in_img
from ttm_image import *

name="zebre2.bmp"
path="../images/%s" % name
img=cv.imread(path,cv.IMREAD_GRAYSCALE )
signal_image=peano_transform_img(img)

w=np.array([0,255])
img_noisy=[]
new_img=[]
erreur=[]

conf=config[2]
signal_noisy_image=bruit_gauss(signal_image,w,conf[0],conf[2],conf[1],conf[3])
(p_est, A_est, m1_est, sig1_est, m2_est, sig2_est)=tt_estim_param_EM_mc(10,signal_noi
sy_image,p,A,conf[0],conf[2],conf[1],conf[3])
print(A_est)

name="beee2.bmp"
path="../images/%s" % name
img=cv.imread(path,cv.IMREAD_GRAYSCALE )
signal_image=peano_transform_img(img)

w=np.array([0,255])
img_noisy=[]
new_img=[]
erreur=[]

conf=config[2]
signal_noisy_image=bruit_gauss(signal_image,w,conf[0],conf[2],conf[1],conf[3])
(p_est, A_est, m1_est, sig1_est, m2_est, sig2_est)=tt_estim_param_EM_mc(10,signal_noi
sy_image,p,A,conf[0],conf[2],conf[1],conf[3])
print(A_est)
```

```
[[0.89916046 0.10083954]
 [0.10609268 0.89390732]]
[[0.99840103 0.00159897]
 [0.00393491 0.99606509]]
```

In [ ]: