Travaux pratiques 6

Images noir & blanc

Informatique tronc commun MPSI et PCSI

Une image est représentée dans un ordinateur sous la forme d'une mosaïque de petits carrés appelés `pixels` (**pic**ture **el**ements). Ces éléments sont codés dans un tableau dont les lignes et les colonnes correspondent aux positions dans l'image. On peut remarquer que cette décomposition en points est aussi celle qu'utilise notre œil qui reçoit la lumière dans des éléments appelés bâtonnets dans la rétine. Dans ce T.P. nous allons nous cantonner au cas des images en niveaux de gris, mais Python gère aussi les images en couleur.

I Images comme représentations de valeurs

Un tableau à deux dimensions sera codé sous la forme d'une liste de listes.
La création d'un tel tableau n'est pas simple. Pour créer un tableau de n lignes avec p colonnes, la commande `I = [[0]*p]*n` produit n lignes qui seront toujours égales ; toute modification sur une ligne sera répercutée sur toutes les autres. On utilisera plutôt

```
I  = [[0]*p for i in range(n)]
```

Cela produit une liste formée de n listes, chacune de taille p.

Les conventions prises par le module `matplotlib` qui assure l'interface entre les données, `I` par exemple, et les représentations graphiques sont les suivantes :

- la liste principale est la liste des lignes, du haut vers le bas,
- le nombre de lignes est `len(I)`,
- chaque élément de cette liste est la liste des valeur d'une ligne de gauche à droite
- le nombre de colonnes, le nombre de pixels par ligne, est `len(I[0])`,
- on accède au pixel d'indice de ligne i et d'indice de colonne j par `I[i][j]`,
- l'échelle utilisée va de la valeur minimale vers la valeur maximale, `matplotlib` s'adapte aux valeurs de la liste de listes.

Par exemple le tableau de valeurs ci-contre sera codé par

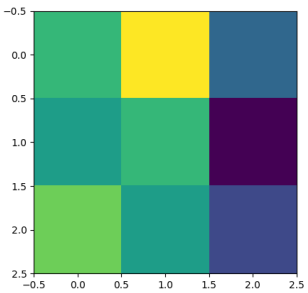
4	7	1
3	4	-2
5	3	0

```
I1 = [[4, 7, 1], [3, 4, -2], [5, 3, 0]]
```

`I1[2][0]` vaut 5

La fonction `imshow` de `matplotlib` représente donc les données sur une échelle de -2 à 7. Elle emploie son propre code de couleurs.

```
import matplotlib.pyplot as plt
plt.imshow(I0)
plt.show()
```



- Sur les anciennes versions de `matplotlib`, il se peut que l'image soit floutée, cela est du au fait que la représentation adoucit les transitions. On supprime cet effet avec la variable optionnelle `interpolation = 'none'` dans `plt.imshow`.
- On peut voir l'échelle des couleurs en ajoutant la commande `plt.colorbar()`
- Il y a beaucoup d"échelles possibles. On peut les voir sur la page

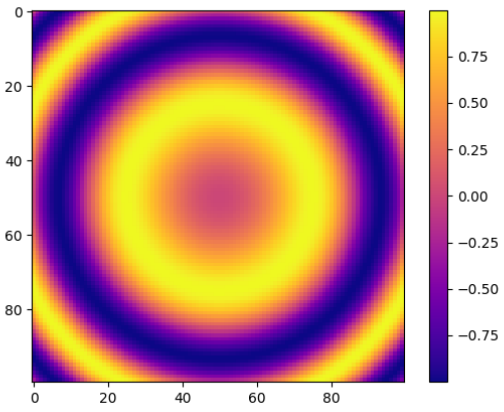
  `https://matplotlib.org/stable/gallery/color/colormap_reference.html`

- On changera l'échelle des couleurs avec la variable optionnelle `interpolation = 'winter'` dans `plt.imshow`, `winter` est un exemple.
  On notera que le nom de l'échelle est passé sous forme d'une chaîne de caractères.

`imshow` est en fait destiné à fournir une représentation d'une fonction de deux variables.

```
import math as m

I2 = [[0]*100 for _ in range(100)]
for i in range(100):
    for j in range(100):
        x = (i-50)/20
        y = (j-50)/20
        I2[i][j] = m.sin(x*x+y*y)
plt.imshow(I2, cmap = "plasma")
plt.colorbar()
plt.show()
```



II Images

Nous allons détourner l'outil `matplotlib` pour afficher des images en niveaux de gris.

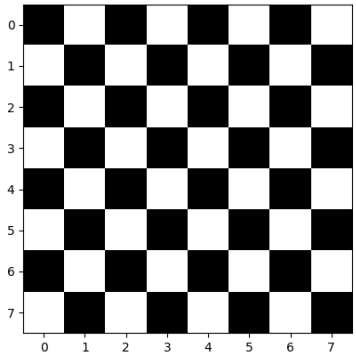
- On emploiera l'échelle de couleurs `gray` : le minimum est noir, le maximum est blanc.
  On peut remplacer la variable optionnelle par la commande `plt.gray()`
- Pour souci de cohérence on imposera des valeurs réelles comprises entre 0 et 1.
- On peut voir sur la dernière image un effet de pixellisation. On peut l'adoucir avec la variable optionnelle `interpolation = 'bilinear'` dans `plt.imshow`.

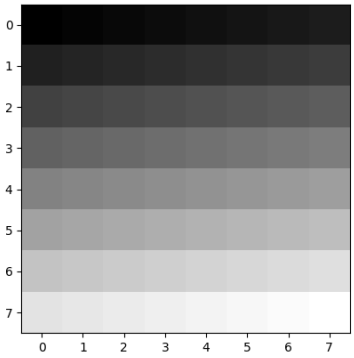
II.1 Images créées

Exercice 1 - Damier
Écrire une fonction d'en-tête `damier(n)` qui affiche un damier de taille $n \times n$.
On pourra utiliser la fonction modulo : n est pair si et seulement si on a `n%2 == 0`.
On n'utilisera pas d'interpolation.

Exercice 2 -
Écrire une fonction d'en-tête `nuance(n)` qui affiche un damier de taille $n \times n$ avec un dégradé de gris à pas constant le long de chaque ligne.

On peut sauvegarder sa création avec `plt.imsave("nom_du_fichier.png", "nom_du_tableau")` ou en cliquant sur l'icône de sauvegarde (une disquette).



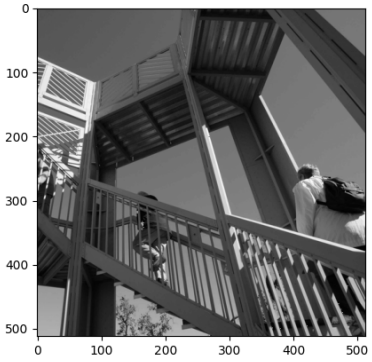


II.2 Image fournie

Dans `scipy.misc` on peut trouver un exemple de photo. Comme ses valeurs sont des entiers de 0 à 255 (8 bits), on les divise par 255.

```
from scipy.misc import ascent

orig = ascent()
n = len(orig)
p = len(orig[0])
I3 = [[0]*p for _ in range(n)]
for i in range(n):
    for j in range(p):
        I3[i][j] = orig[i][j]/255
plt.gray()
plt.imshow(I3)
plt.show()
```



On peut remplacer la copie par

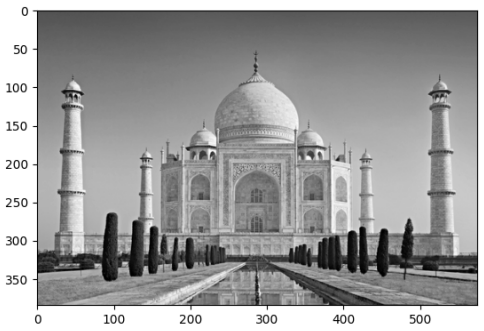
```
I3 = [[orig[i][j]/255 for j in range(p)] for i in range(n)]
```

II.3 Image chargée

`matplotlib` permet de lire des images en format `png` et les transforme en matrice (ce seront des `np.array` de `numpy` que nous utiliserons comme des listes Python).

Le dossier `public` devrait contenir une photo du Taj-Mahal.

```
I4 = plt.imread('taj-NB.png')
plt.gray()
plt.imshow(I4)
plt.show()
```



III Transformations

III.1 Transformations géométriques

On veut écrire une fonction qui affiche une image transformée par symétrie axiale. Son axe sera vertical et passera par le centre de l'image.

Exercice 3 - Symétrie verticale
Soit une matrice de taille $n \times p$ représentant une image. Quelle est l'image du point de coordonnées (i, j) par cette transformation axiale ?
Compléter la fonction `symetrie(image)` pour afficher l'image transformée de `image`.

```
def symetrieV(image) :
    n = len(image)
    p = len(image[0])
    imageSym = [[0]*p for _ in range(n)]
    for i in range(n) :
        for j in range(p) :
            imageSym[i][j] = ..........
    plt.gray()
    plt.imshow(imageSym)
    plt.show()
```

Exercice 4 - Symétrie horizontale
Écrire de même une fonction qui affiche une image renversée selon l'axe horizontal passant par le centre

Exercice 5 - Symétrie centrale
Écrire une fonction qui affiche une image transformée par symétrie centrale. Son centre sera le centre de l'image.

On veut écrire une fonction qui réduit la taille de l'image pour obtenir 4 images analogues en apparence positionnées comme sur une planche de photomaton. L'image finale a le même nombre de lignes et de colonnes que l'originale.
Pour que cette transformation soit bijective, on découpe l'image initiale en paquets carrés de quatre pixels (2) puis pour chaque paquet carré de quatre pixels, on utilise celui en haut à gauche pour l'image réduite en haut à gauche, celui en haut à droite pour l'image réduite en haut à droite, ...On supposera que les tailles de l'image sont paires

Exercice 6 - Symétrie horizontale
Écrire la fonction correspondante.

III.2 Transformations locales

On va ici appliquer une fonction sur chaque pixel. f une fonction de $[0;1]$ dans lui-même et on va copier l'image dans un autre tableau de même taille avec la valeur `f(I[i][j])` aux indices i et j.

```
def modifie(image, a, b, ...) :
    n = len(image)
    p = len(image[0])
    copie = [[0]*p for _ in range(n)]
    for i in range(n) :
        for j in range(p) :
            copie[i][j] = f(image[i][j], a, b, ...)
    plt.gray()
    plt.imshow(copie)
    plt.show()
```

a, b, ...sont des paramètres supplémentaires de f

Exercice 7 - Seuillage
Écrire une fonction `seuil(x, h)` qui reçoit deux réels x et h et qui renvoie 0 dans la cas $x < h$ et 1 sinon.
En déduire une fonction `seuil_image(image, h)` qui affiche l'image transformée par f.

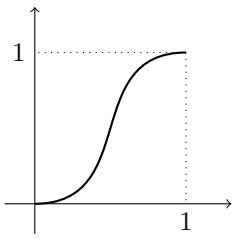
On peut généraliser en calculant plusieurs valeurs de seuil : on obtient un effet de sérigraphie.
La fonction $x \mapsto \frac{1}{n}\lfloor nx \rfloor$ ne donnera que les valeurs 0, $\frac{1}{n}$, $\frac{2}{n}$, ..., $\frac{n-1}{n}$, 1.
Pour faire psychédélique, on pourra utiliser `cmap = "gist_rainbow"`.

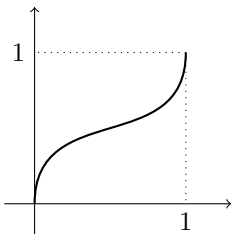
Exercice 8 - Postérisation
Écrire une fonction `annees60(image, n)` qui qui affiche l'image dont les pixels sont discrétisés en prenant n valeurs.

On peut aussi modifier les valeurs en les orientant vers le blanc ou le noir. Il suffit de trouver une fonction de $[0;1]$ dans lui même qui vérifie $f(x) \geqslant x$ pour donner plus de luminosité ou $f(x) \leqslant x$ pour assombrir.
Les fonctions $x \mapsto x^a$ sont des bons candidats : $a > 1$ assombrit, $a < 1$ éclaircit.

Exercice 9 - Luminosité
Écrire une fonction `luminosite(image, a)` qui qui affiche l'image modifiée par cette mé thode.

Plutôt que déplacer la luminosité on peut modifier la différentiation entre le noir et le blanc : le contraste. Augmenter le contraste consiste à diminuer les petites valeurs et augmenter les grandes.
Les fonctions que l'on recherche auront un graphe qui ressemble aux graphes suivant





Exercice 10 - Luminosité
Écrire des fonctions `contrastePlus(image)` et `contrasteMoins(image)` qui reçoivent une image et qui renvoient une nouvelle image dont le contraste a été augmenté ou diminué.

III.3 Usage des points voisins

Dans cette partie nous allons modifier les images de manière plus globale : un point sera modifié en fonction de son environnement. Comme les valeurs dépendent des voisins les transformations ne seront pas possibles sur les bords ; on choisira de les laisser tels quels ou de les supprimer.

Exercice 11 - Floutage
Pour calculer une image floutée on peut remplacer chaque point par la moyenne de ses 25 voisins (de $i-2$ à $i+2$ et de $j-2$ à $j+2$).

Exercice 12 - Netteté
On peut, inversement, augmenter le "piqué" de la photo en augmentant le contraste local. On pourra, dans ce but, remplacer chaque valeur `im[i][j]` par
`5*im[i][j] - im[i-1][j] - im[i][j-1] - im[i+1][j] - im[i][j+1]`.

Il y a des petits défauts dans la photo qui donnent de grandes valeurs indésirables au résultat. On pourra tronquer les valeurs en remplaçant les négatifs par 0 et les valeurs supérieures à 1 par 1.

Exercice 13 - Contour
Qu'obtient-on si on remplace chaque valeur `im[i][j]` par
`4*im[i][j] - im[i-1][j] - im[i][j-1] - im[i+1][j] - im[i][j+1]` ?

IV Exemples de résultats

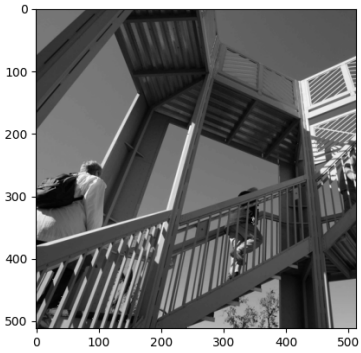


FIGURE 1 – Exercice 3

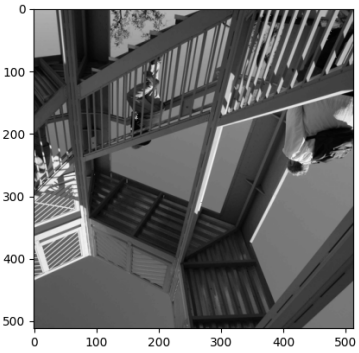


FIGURE 2 – Exercice 4

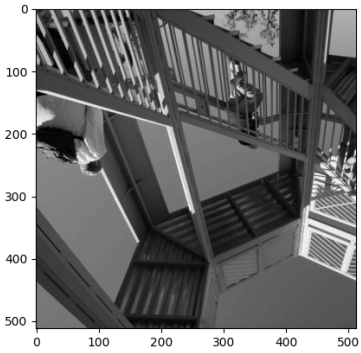


FIGURE 3 – Exercice 5

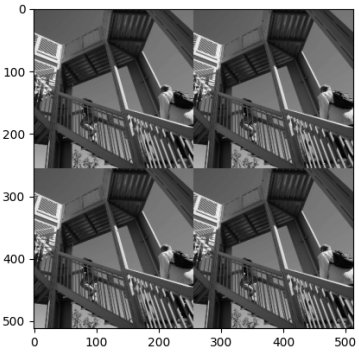


FIGURE 4 – Exercice 6

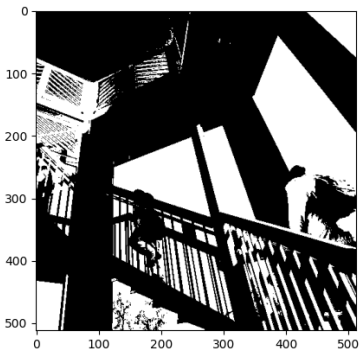


FIGURE 5 – Exercice 7, seuil $= 0,4$

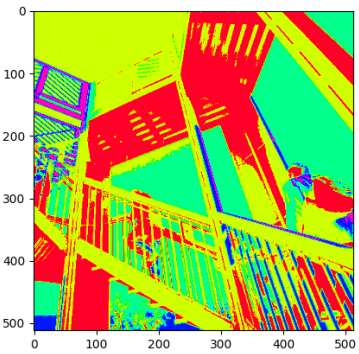


FIGURE 6 – Exercice 8, $n = 4$

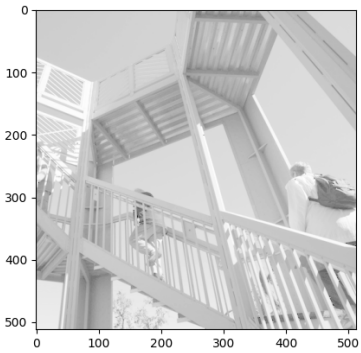


FIGURE 7 – Exercice 9, $a = 0,2$

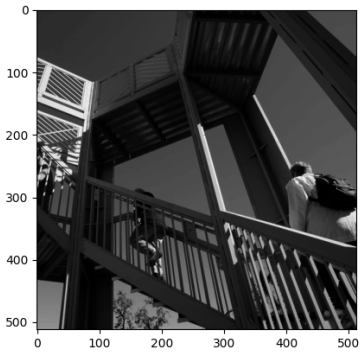


FIGURE 8 – Exercice 9, $a = 1,6$

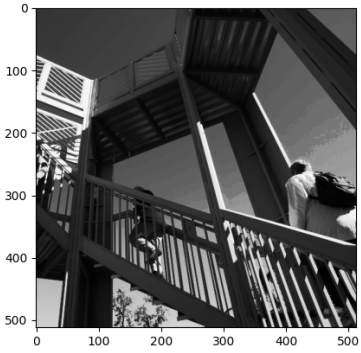


FIGURE 9 – Exercice 10

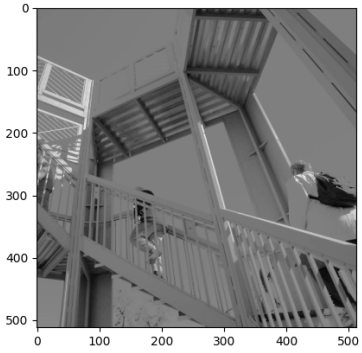


FIGURE 10 – Exercice 10

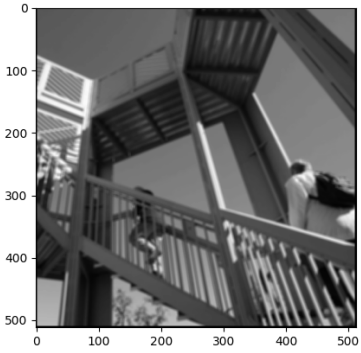


FIGURE 11 – Exercice 11

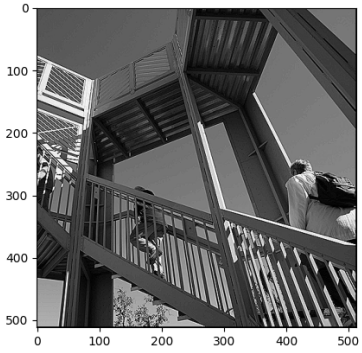


FIGURE 12 – Exercice 12

Solutions

Solution de l'exercice 1 - Damier

```
def damier(n) :
    d = [[0]*n for _ in range(n)]
    for i in range(n) :
        for j in range(n) :
            d[i][j] = (i + j)%2
    plt.gray()
    plt.imshow(d)
    plt.show()
```

Solution de l'exercice 2 -

```
def nuance(n) :
    nua = [[0]*n for _ in range(n)]
    for i in range(n) :
        for j in range(n) :
            nua[i][j] = (i*n + j)/(n*n-1)
    plt.gray()
    plt.imshow(nua)
    plt.show()
```

Solution de l'exercice 3 - Symétrie verticale

```
def symetrieV(image) :
    n = len(image)
    p = len(image[0])
    imageSym = [[0]*p for _ in range(n)]
    for i in range(n) :
        for j in range(p) :
            imageSym[i][j] = image[i][p-1-j]
    plt.imshow(imageSym)
    plt.show()
```

Solution de l'exercice 4 - Symétrie horizontale

```
def symetrieH(image) :
    n = len(image)
    p = len(image[0])
    imageSym = [[0]*p for _ in range(n)]
    for i in range(n) :
        for j in range(p) :
            imageSym[i][j] = image[n-1-i][j]
    plt.gray()
    plt.imshow(imageSym)
    plt.show()
```

Solution de l'exercice 5 - Symétrie centrale
C'est la composée des deux fonctions précédentes

```
def symetrieC(image) :
    n = len(image)
    p = len(image[0])
    imageSym = [[0]*p for _ in range(n)]
    for i in range(n) :
        for j in range(p) :
            imageSym[i][j] = image[n-1-i][p-1-j]
    plt.gray()
    plt.imshow(imageSym)
    plt.show()
```

Solution de l'exercice 6 - Symétrie horizontale

```
def photomaton(image) :
    n = len(image)
    p = len(image[0])
    pm = [[0]*p for _ in range(n)]
    for i in range(n//2) :
        for j in range(p//2) :
            pm[i][j] = image[2*i][2*j]
            pm[i + n//2][j] = image[2*i+1][2*j]
            pm[i][j + p//2] = image[2*i][2*j+1]
            pm[i + n//2][j + p//2] = image[2*i+1][2*j+1]
    plt.gray()
    plt.imshow(pm)
    plt.show()
```

Solution de l'exercice 7 - Seuillage

```
def seuil(x, h):
    if x < h:
        return 0
    else:
        return 1

def seuil_image(image, h):
    n = len(image)
    p = len(image[0])
    copie = [[0]*p for _ in range(n)]
    for i in range(n) :
        for j in range(p) :
            copie[i][j] = seuil(image[i][j], h)
    plt.gray()
    plt.imshow(copie)
    plt.show()
```

Solution de l'exercice 8 - Postérisation

```
def discretiser(x, n):
    return m .floor(n*x)/n

def annees60(image, n):
    n = len(image)
    p = len(image[0])
    copie = [[0]*p for _ in range(n)]
    for i in range(n) :
        for j in range(p) :
            copie[i][j] = discretiser(image[i][j], n)
    plt.imshow(copie, cmap = "gist_rainbow")
    plt.show()
```

Solution de l'exercice 9 - Luminosité

```
def luminosité(image, a):
    n = len(image)
    p = len(image[0])
    copie = [[0]*p for _ in range(n)]
    for i in range(n) :
        for j in range(p) :
            copie[i][j] = image[i,j]**a
    plt.gray()
    plt.imshow(copie)
    plt.show()
```

Solution de l'exercice 10 - Luminosité
On peut faire intervenir un paramètre

```
def f(x, a):
    if  x < 0.5:
        return (1 - (1-2*x)**a)/2
    else:
        return (1 + (2*x-1)**a)/2

def contraste(image, a):
    n = len(image)
    p = len(image[0])
    copie = [[0]*p for _ in range(n)]
    for i in range(n) :
        for j in range(p):
            copie[i][j] = f(image[i][j], a)
    plt.gray()
    plt.imshow(copie)
    plt.show()
```

$a = 0.5$ augmente le contraste, $a = 2$ le diminue.

Solution de l'exercice 11 - Floutage

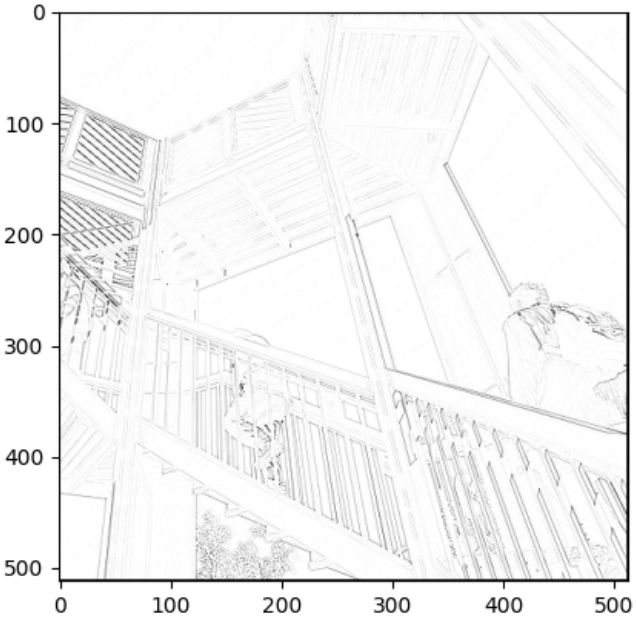
```
def flou(image):
    n = len(image)
    p = len(image[0])
    copie = [[0]*p for _ in range(n)]
    for i in range(2, n-2) :
        for j in range(2, p-2) :
            for ii in range(i-2, i+3):
                for jj in range(j-2, j+3):
                    copie[i][j] += image[ii][jj]/25
    plt.gray()
    plt.imshow(copie)
    plt.show()
```

Solution de l'exercice 12 - Netteté

```
def nette(image):
    n = len(image)
    p = len(image[0])
    copie = [[0]*p for _ in range(n)]
    for i in range(1, n-1) :
        for j in range(1, p-1) :
            copie[i][j] = 5*image[i][j] - image[i-1][j] - image[i
               ][j-1] - image[i+1][j] - image[i][j+1]
            if copie[i][j] < 0:
                copie[i][j] = 0
            if copie[i][j] > 1:
                copie[i][j] = 1
    plt.gray()
    plt.imshow(copie)
    plt.show()
```

Solution de l'exercice 13 - Contour
On inverse l'image, on obtient le contour des objets.

```
def contour(image):
    n = len(image)
    p = len(image[0])
    copie = [[0]*p for _ in range(n)]
    for i in range(1, n-1) :
        for j in range(1, p-1) :
            copie[i][j] = 1 - 4*image[i][j] + image[i-1][j] +
               image[i][j-1] + image[i+1][j] + image[i][j+1]
            if copie[i][j] < 0:
                copie[i][j] = 0
            if copie[i][j] > 1:
                copie[i][j] = 1
    plt.gray()
    plt.imshow(copie)
    plt.show()
```


0
100
200
300
400
500
0
100
200
300
400
500