TP 17 : Tableaux à 2 dimensions, images

Exercice 1 *Création d'un tableau de taille $n \times n$.* Pour un entier $n \geqslant 1$, on considère le tableau A de taille n :

$$A = \begin{bmatrix} 1 & 2 & 3 & \cdots & n-1 & n \\ 2 & 1 & 0 & \cdots & 0 & n-1 \\ 3 & 0 & \ddots & \ddots & \vdots & n-2 \\ \vdots & \vdots & \ddots & \ddots & 0 & \vdots \\ n-1 & 0 & \cdots & 0 & 1 & 2 \\ n & n-1 & n-2 & \cdots & 2 & 1 \end{bmatrix}$$

Écrire une fonction **`construire_tableau(n)`** qui construit et renvoie ce tableau A.

Exercice 2 *Minimum des éléments d'un tableau.*

(a) Écrire une fonction **`minimum(A)`** qui détermine et renvoie la valeur minimale des éléments du tableau A à deux dimensions.

(b) Écrire une fonction **`indices_minimum(A)`** qui détermine et renvoie le couple d'indices (i, j) pour lequel $A[i, j]$ est minimal.

Traitement d'images

◇ Vous commencerez par récupérer les quatre fichiers qui ont été envoyés :

```
Ange_au_sourire.png
PapillonAvecBruit.png
Cellules.png
images.py
```

et les placer dans le même répertoire que vos programmes PYTHON. Vous pouvez ouvrir le fichier **`images.py`** dans PYZO et l'exécuter, un image doit s'afficher. Vous réaliserez tous les exercices qui suivent dans ce fichier.

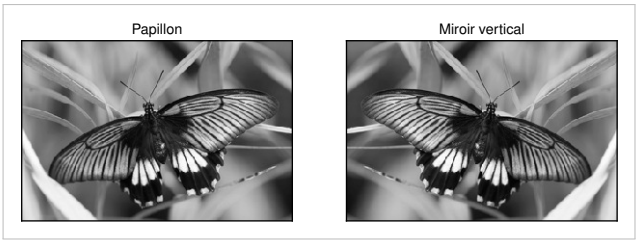
Exercice 3 *Négatif d'une image.* Écrire une fonction **`negatif(A)`** qui construit et renvoie le tableau B, de même taille que A, tel que $B[i, j] = 255 - A[i, j]$. On commencera par écrire dans la fonction **`(n,m) = A.shape`** afin que n représente le nombre de lignes de A et m le nombre de colonnes. On définira ensuite **`B = np.zeros((n,m))`** afin de construire un tableau B de même taille que A et dont tous les coefficients sont nuls. On définira alors $B[i, j]$ avec la relation ci-dessus en utilisant deux boucles. Afficher l'image obtenue en appliquant cette fonction sur les exemples d'images.

Exercice 4 *Transformations géométriques sur une image.*

(a) Écrire une fonction **`miroir_vert(A)`** qui construit et renvoie une nouvelle image B de même taille que A et telle que :

$$B[i,j] = A[i, m-j-1]$$

où m est le nombre de colonnes de A. Par exemple :

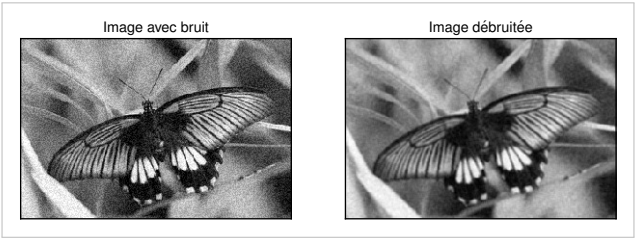


(b) Définir de même la fonction **`miroir_horiz(A)`** :

Exercice 5 *Éliminer un bruit parasite.* Pour atténuer le bruit qui parasite une image A, on peut créer une nouvelle image B telle que $B[i,j]$ soit la moyenne des valeurs de A situées autour de $A[i,j]$.

(a) Écrire une fonction **`moyenne(A,i,j,p)`** qui réalise la moyenne des éléments $A[x,y]$ pour lesquels $i-p \leqslant x \leqslant i+p$ et $j-p \leqslant y \leqslant j+p$. On prendra garde au fait que $i-p$ et $j-p$ peuvent être négatifs et de même $i+p$ et $j+p$ peuvent dépasser le nombre de lignes ou de colonnes de A.

(b) Écrire une fonction **`debruiter(A,p)`** qui construit à partir de l'image A une nouvelle image B obtenue en utilisant la fonction moyenne précédente. Tester avec $p=1$. Comparer avec le résultat obtenu pour $p=2$.



Exercice 6 *Histogramme d'une image et seuillage.* On considère dans cet exercice une image représentée par un tableau A à n lignes et p colonnes dont les éléments sont des entiers compris entre 0 et 255.

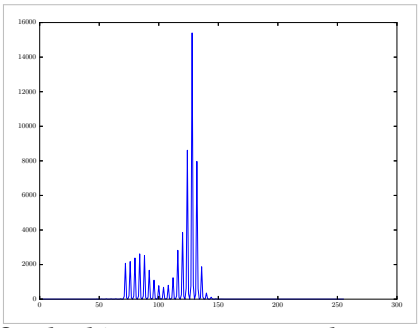
(a) On considère un nombre entier v appelé *seuil*. On définit une nouvelle image B, de même taille que A, en posant :

$$B[i,j] = 0 \quad \text{si} \quad A[i,j] \leqslant v$$
$$= 255 \quad \text{sinon}$$

On dit que B est l'image monochrome obtenue à partir de A par *seuillage*. Écrire une fonction **`seuillage(A,v)`** qui construit et renvoie l'image B ainsi définie.

(b) L'histogramme de l'image A est la liste H de taille 256 telle que $H[i]$ est égal au nombre de cases de valeur i dans A. Écrire une fonction **`histogramme(A)`** qui construit et retourne la liste H. On pourra ensuite effectuer le tracé de l'histogramme avec (par exemple) :

```
plt.plot(histogramme(Cellules))
plt.show()
```



(c) Examiner l'histogramme de l'image **`Cellules.png`** afin de déterminer un seuil pertinent pour effectuer le seuillage.

Corrections

Ex 1. On commence par créer un tableau de taille $n \times n$ dont tous les coefficients sont nuls. On utilise ensuite une boucle *for* pour remplir la première et la dernière ligne, la première et la dernière colonne ainsi que la diagonale.

```
import numpy as np

def construire_tableau(n):
    A = np.zeros((n,n))
    for i in range(1,n+1):
        A[i-1,i-1] = 1 # Diagonale
        A[i-1,0] = i # Première colonne
        A[0,i-1] = i # Première ligne
        A[n-1,i-1] = n+1-i # Dernière colonne
        A[i-1,n-1] = n+1-i # Dernière ligne
    return A

print(construire_tableau(6))
```

```
[[ 1. 2. 3. 4. 5. 6.]
[ 2. 1. 0. 0. 0. 5.]
[ 3. 0. 1. 0. 0. 4.]
[ 4. 0. 0. 1. 0. 3.]
[ 5. 0. 0. 0. 1. 2.]
[ 6. 5. 4. 3. 2. 1.]]
```

Ex 2. Le principe est le même que pour déterminer le minimum d'une liste mais il faut faire deux boucles imbriquées.

```
def minimum(A):
    (n,p) = A.shape
    m = A[0,0] # La valeur minimale rencontrée jusqu'à présent
    for i in range(0,n): # Indice de ligne entre 0 et n-1
        for j in range(0,p): # Indice de colonne entre 0 et p-1
            if A[i,j]<m:
                m = A[i,j]
    return m

print(minimum(np.array([[1,6],[-2,4]])))
```

```
def indices_minimum(A):
    (n,p) = A.shape
    im = 0 # Indice de ligne de la valeur minimale rencontrée jusqu'à présent
    jm = 0 # Indice de colonne de la valeur minimale rencontrée jusqu'à présen
    for i in range(0,n): # Indice de ligne entre 0 et n-1
        for j in range(0,p): # Indice de colonne entre 0 et p-1
            if A[i,j]<A[im,jm]:
                im = i
                jm = j
    return (im,jm)

print(indices_minimum(np.array([[1,6],[-2,4]])))
```

```
(1, 0)
```

Ex 3.

/home/ba/Enseignement/PC/Python/images.py

```
from scipy.misc import imread, imsave
import numpy as np
import matplotlib.pyplot as plt

# Fonctions d'affichage

def afficher1image(A):
        plt.xticks([])
        plt.yticks([])
        plt.imshow(A,cmap=plt.cm.gray,interpolation='nearest',vmin=0,vmax=255)
        plt.show()

def afficher2images(A,B):
        plt.subplot(1,2,1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(A,cmap=plt.cm.gray,interpolation='nearest',vmin=0,vmax=255)
        plt.subplot(1,2,2)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(B,cmap=plt.cm.gray,interpolation='nearest',vmin=0,vmax=255)
        plt.show()

# Lecture des fichiers image

Ange = imread('Data/Ange_au_sourire.png')
Papillon = imread('Data/Papillon.png')
PapillonBruit = imread('Data/PapillonAvecBruit.png')
Cellules = imread('Data/Cellules.png')

# Les fonctions de manipulation d'images
```

```
def negatif(A):
        (n,p) = A.shape
        B = np.zeros((n,p))
        for i in range(0,n):
                for j in range(0,p):
                        B[i,j] = 255-A[i,j]
        return B

def miroir_vert(A):
        (n,p) = A.shape
        B = np.zeros((n,p))
        for i in range(0,n):
                for j in range(0,p):
                        B[i,j] = A[i,p-j-1]
        return B

def miroir_horiz(A):
        (n,p) = A.shape
        B = np.zeros((n,p))
        for i in range(0,n):
                for j in range(0,p):
                        B[i,j] = A[n-i-1,j]
        return B

def moyenne(A,i,j,p):
        (n,m) = A.shape
        s = 0 # La somme des éléments situés autour de A[i,j]
        c = 0 # Le nombre d'éléments situés autour de A[i,j]
        for x in range(i-p,i+p+1):
                for y in range(j-p,j+p+1):
                        if x>=0 and x<n and y>=0 and y<m:
                                s = s+A[x,y]
                                c = c+1
        return s/c

def debruiter(A,p):
        (n,m) = A.shape
        B = np.zeros((n,m))
        for i in range(0,n):
                for j in range(0,m):
                        B[i,j] = moyenne(A,i,j,p)
        return B

def histogramme(A):
        (n,m) = A.shape
        H = [0]*256
        for i in range(0,n):
                for j in range(0,m):
                        H[A[i,j]] = H[A[i,j]]+1
```

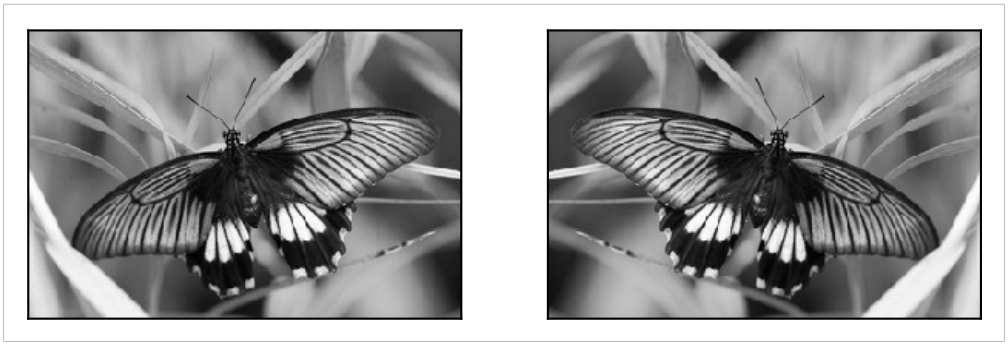
```
        return H

def seuillage(A,v):
        (n,m) = A.shape
        B = np.zeros((n,m))
        for i in range(0,n):
                for j in range(0,m):
                        if A[i,j]<=v:
                                B[i,j] = 0
                        else:
                                B[i,j] = 255
        return B
```

```
afficher2images(Papillon,negatif(Papillon))
```

Ex 4.

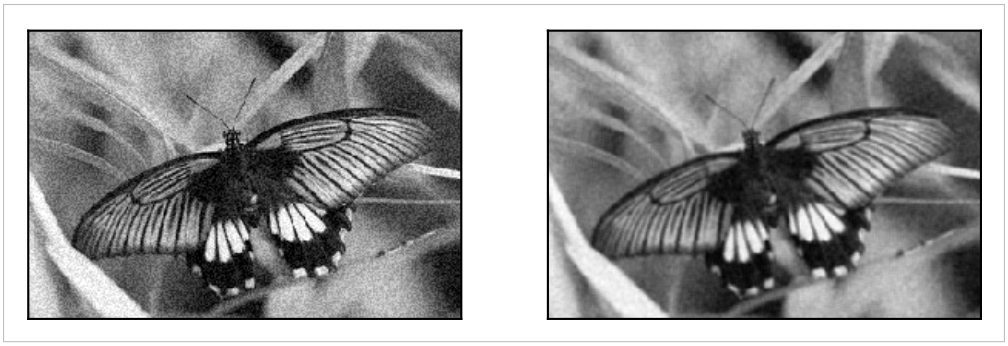
```
afficher2images(Papillon,miroir_vert(Papillon))
```



```
afficher2images(Papillon,miroir_horiz(Papillon))
```

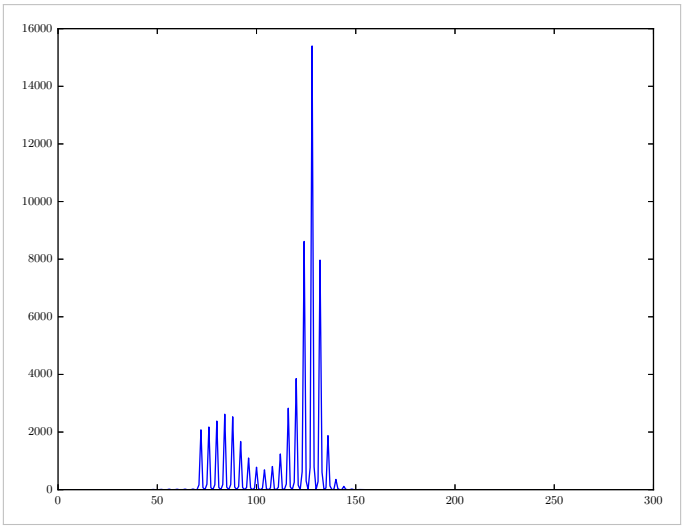
Ex 5.

```
afficher2images(PapillonBruit,debruiter(PapillonBruit,1))
```



Ex 6.

```
plt.plot(histogramme(Cellules))
```



L'histogramme semble montrer deux groupes différents, approximativement séparés par le niveau 100.

```
afficher2images(Cellules,seuillage(Cellules,100))
```