

EXAMEN DE FIN DE TRIMESTRE 2**Matière : INFORMATIQUE****Classes : 2^{ème} Année Durée : 1h30****Préparation : MP, PC, PT et BG****DOCUMENTS NON AUTORISES****EXERCICE 1**

Une image numérique peut être représentée par une matrice dans laquelle chaque case représente un pixel (unité minimale adressable par le contrôleur vidéo et supposé de couleur uniforme). Le *format* de l'image (png, jpeg, etc) désigne la manière dont cette matrice est représentée en mémoire.

- Pour une image en noir et blanc, chaque pixel ne peut prendre que deux valeurs, ce qui permet de l'encoder sur un seul bit ;
- pour une image en niveau de gris, chaque teinte de gris (en passant du noir au blanc) est encodée sur un octet, ce qui permet de représenter $2^8 = 256$ nuances de gris;
- pour une image en couleur, chaque pixel est encodé sur trois octets, chaque octet représentant une nuance de couleur. Par exemple, dans le cas du format RGB (*Red, Green, Blue*), le premier octet dose la quantité de rouge, le second la quantité de vert et le troisième la quantité de bleu pour une synthèse additive ;
- enfin, les pixels d'une image en couleur peuvent être encodés par quatre octets au format RGBA (*A pour alpha*), le quatrième octet codant la transparence du pixel.

Manipulation d'image en PYTHON

Durant cette EPREUVE nous aurons besoin de trois modules : NUMPY OU SCIPY pour la manipulation des matrices, MATPLOTLIB.PY PLOT pour la visualisation des images et enfin IMAGEIO pour la conversion image/matrice. Nous aurons besoin aussi des fonctions suivantes:

- La fonction `io.imread(source)` permet de convertir une image au format courant (png, jpeg, etc) en un tableau NUMPY. Elle prend en argument une chaîne de caractère décrivant un fichier présent sur votre ordinateur ou l'adresse http d'une image présente sur le net.
- La fonction `io.imwrite(cible, tab)` permet au contraire de convertir un tableau NUMPY (l'argument *tab*) en un fichier image (décrit par la chaîne de caractères *cible*).
- La fonction `plt.imshow(tab)` permet de visualiser l'image décrite par un tableau. À noter, pour visualiser une image en niveau de gris il faut préciser l'échelle chromatique à utiliser, ici : `plt.imshow(tab, cmap= ' gray ')`.

Question 0.

Ecrire les instructions permettant :

- d'importer les modules nécessaires pour la manipulation des matrices
- de récupérer l'image 'picasso.png' qui nous servira de modèle, la visualiser puis l'enregistrer dans le répertoire courant.

Ainsi définie `im` est tableau NUMPY dont les éléments sont de type `np.uint8` (entiers non signés représentés sur 8 bits, autrement dit un entier compris entre 0 et 255). La méthode `shape` donne les dimensions de ce tableau : `im.shape = (n,p)` pour une image en niveau de gris, `im.shape = (n,p,3)` pour une image couleur, `im.shape = (n,p,4)` pour une image avec composante alpha. L'entier n est le nombre de lignes de l'image et p le nombre de colonnes, le pixel de coordonnées (0,0) est situé en haut à gauche de l'image.

Remarque. En réalité, le format de l'objet importé par la commande `io.imread` est plus complexe qu'un simple tableau NUMPY (il contient des informations complémentaires sur le contenu de l'image) et la manipulation de ces images s'en trouve ralentie. Pour accélérer la vitesse d'exécution des fonctions que vous aurez à écrire, vous aurez intérêt à transformer ces dernières en simples tableaux NUMPY en rajoutant au code précédent la commande :

```
im = np.array(im)
```

Question 1.

Rédiger une fonction **symetrie** qui prend en argument une image et retourne une nouvelle image, obtenue par symétrie autour d'un axe vertical passant par le milieu de l'image.

Indication : la fonction `np.zeros_like(m)` crée un tableau nul de même type et de mêmes dimensions que le tableau `m`.

Question 2.

Rédiger une fonction **rotation** qui prend en argument une image et retourne une nouvelle image, obtenue par rotation d'un angle $n/2$ autour du centre de l'image.

Indication : la fonction `np.zeros((n, p, s), dtype=np.uint8)` crée un tableau vide de dimensions $n \times p \times s$ dont les éléments sont de type `np.uint8`.

Question 3.

Rédiger une fonction **negatif** qui prend en argument une image et retourne son négatif, c'est-à-dire l'image dans laquelle chaque composante de couleur de chaque pixel est remplacée par sa valeur complémentaire dans l'intervalle [0,255].

Question 4. Conversion en niveau de gris

Pour convertir une image couleur en niveau de gris, un pixel représenté par ses composantes (r, g, b) doit être remplacé par un pixel à une seule composante $y \in [0,255]$ appelée *luminance*. Cette quantité, qui traduit la sensation visuelle de luminosité, dépend de manière inégale des trois composantes RGB. La formule communément recommandée pour cette conversion est la suivante :

$$y = 0,2126r + 0,7152g + 0,0722b \quad (y \text{ étant arrondi à l'entier le plus proche})$$

(pour un œil humain le vert paraît plus lumineux que le rouge, lui-même plus lumineux que le bleu).

Rédiger une fonction **niveaudegris** qui prend en argument une image couleur et retourne une nouvelle image convertie en niveau de gris.

Traitement d'image

La plupart des filtres de traitement d'images utilisent une convolution par un *masque* pour réaliser une modification. Ces masques sont des matrices de petites tailles, en général 3×3 (taille que nous allons considérer par la suite) ou 5×5 :

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

Le filtre associé à ce masque transforme la matrice $M = (m_{ij})$, associée à une certaine image, en la matrice $M \otimes C = (m^*_{ij})$ par une opération appelée *produit de convolution* et définie par la relation : $\forall i, j$,

$$m^*_{ij} = c_{11}m_{i-1,j-1} + c_{12}m_{i-1,j} + c_{13}m_{i-1,j+1} + c_{21}m_{i,j-1} + c_{22}m_{i,j} + c_{23}m_{i,j+1} + c_{31}m_{i+1,j-1} + c_{32}m_{i+1,j} + c_{33}m_{i+1,j+1}$$

Dans le cas d'une image en couleur, on peut choisir d'appliquer le même masque à chacune des trois composantes RGB de l'image ou leur appliquer des masques différenciés, en fonction de l'effet désiré. Dans la suite de ce sujet, et dans un but simplificateur, nous appliquerons le même masque à chacune des trois composantes de couleur.

Remarque. Lorsque le pixel initial est sur un bord, une partie de la convolution porte en dehors des limites de l'image. Là encore, nous conviendrons pour simplifier de laisser inchangés ces pixels.

Question 5.

Rédiger une fonction **convolution** qui prend en arguments deux matrices M (associée à une image) et C et retourne la matrice $M \otimes C$. Les éléments de la matrice C seront *a priori* de type *float* tandis que ceux des matrices M et $M \otimes C$ seront de type *np.uint8*. Il conviendra donc, lorsque $m^*_{ij} < 0$ ou $m^*_{ij} > 255$ de remplacer les valeurs calculées par respectivement 0 et 255. Ne pas oublier non plus que les pixels d'une image en couleur possèdent trois composantes et que chacune de ces composantes doit être traitée.

Exemples de masques

Lissage Pour obtenir un effet de lissage on utilise le masque C_1 : chaque pixel est remplacé par la moyenne de lui-même et de ses huit voisins ; pour cette raison on parle de filtre *moyenneur*.

Augmentation du contraste Au contraire, pour augmenter le contraste on utilise le masque C_2 .

Repoussage Enfin, le masque C_3 donne un effet de relief à l'image, appelé *repoussage*.

$$C_1 = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad C_2 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad C_3 = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

FIGURE 1 – Trois filtres classiques en traitement d'images.

Question 6.

Rédiger trois fonctions **lissage**, **contraste** et **repoussage** qui prennent toutes trois en argument une image et retournent une nouvelle image obtenue en appliquant respectivement un effet de lissage, d'augmentation de contraste et de repoussage, et observer le résultat sur l'image test.

EXERCICE 2

L'objectif de l'exercice est l'utilisation de la méthode des moindres carrés pour approximer un polynôme à partir d'observations. La spécificité de cette méthode est de minimiser la somme des distances entre un polynôme g approximant et n points expérimentaux.

Etant donné une observation de n points distincts $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, on cherche à approcher cette observation, au sens des moindres carrés, par un polynôme $g(x) = a_0 + a_1x + a_2x^2 + \dots + a_mx^m$ de degré m égal

à $n-1$, donnant une meilleure approximation des valeurs y_i . Ceci revient à minimiser la distance qui sépare un point expérimental (x_i, y_i) du point approximant $(x_i, g(x_i))$.

Le polynôme donnant la meilleure approximation est celui qui minimise la somme S , des écarts entre les y_i et les $g(x_i)$, donnée par la formule suivante :

$$S(a_0, a_1, a_2, \dots, a_m) = \sum_{i=1}^n |y_i - (a_0 + a_1 x_i + a_2 x_i^2 + \dots + a_m x_i^m)|$$

Afin d'identifier la distance moyenne minimale, on cherche l'ensemble des valeurs des paramètres a_i minimisant cette somme.

Ce qui conduit à résoudre le système linéaire suivant :

$$\begin{bmatrix} \sum_{i=1}^n x_i^0 & \sum_{i=1}^n x_i^1 & \dots & \sum_{i=1}^n x_i^m \\ \sum_{i=1}^n x_i^1 & \sum_{i=1}^n x_i^2 & \dots & \sum_{i=1}^n x_i^{m+1} \\ \vdots & \vdots & \dots & \vdots \\ \sum_{i=1}^n x_i^{m-1} & \sum_{i=1}^n x_i^m & \dots & \sum_{i=1}^n x_i^{2m-1} \\ \sum_{i=1}^n x_i^m & \sum_{i=1}^n x_i^{m+1} & \dots & \sum_{i=1}^n x_i^{2m} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \\ a_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n x_i^0 y_i \\ \sum_{i=1}^n x_i^1 y_i \\ \vdots \\ \sum_{i=1}^n x_i^{m-1} y_i \\ \sum_{i=1}^n x_i^m y_i \end{bmatrix}$$

En posant $U_k = \sum_{i=1}^n x_i^k$ et $v_k = \sum_{i=1}^n x_i^k y_i$, le système s'écrit alors:

$$\begin{bmatrix} U_0 & U_1 & \dots & U_m \\ U_1 & U_2 & \dots & U_{m+1} \\ \vdots & \vdots & \dots & \vdots \\ U_{m-1} & U_m & \dots & U_{2m-1} \\ U_m & U_{m+1} & \dots & U_{2m} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \\ a_m \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{m-1} \\ v_m \end{bmatrix} \Rightarrow U \cdot a = v$$

Travail demandé :

Dans la suite, on suppose que les n points représentant une observation, sont stockés dans une liste de tuples Lp , où chaque tuple représente les coordonnées d'un point.

Le travail demandé consiste à déterminer les coefficients a_i du polynôme d'interpolation $g(x)$.

1. Ecrire une fonction python, nommée **puiss**, qui, pour un réel x et un entier p , crée et retourne la liste $[x^0, x^1, \dots, x^{2p}]$.
2. Ecrire une fonction python, nommée **list_puiss**, qui, à partir d'une liste de réels L et d'un entier p , crée et retourne une liste de listes contenant, pour chaque réel r de L , une liste $[r^0, r^1, \dots, r^{2p}]$.
3. Ecrire une fonction python, nommée **calcul_mat**, qui, à partir de la liste de points Lp , crée et retourne la matrice U .
4. Ecrire une fonction python, nommée **calcul_vect**, qui à partir de la liste de points Lp , crée et retourne le vecteur v .