TP : Images

Une image bitmap (anglicisme), ou image matricielle, est un tableau de points colorés. La justaposition de ces points constitue une image.

I) Images en niveau de gris

A) Premiers pas

Chargez Numpy (pour manipuler des tableaux) et Matplotlib.pyplot (pour l'affichage graphique). La dernière ligne est ad hoc et sera expliquée plus tard.

```
import matplotlib.pyplot as plt
import numpy as np
plt.set_cmap('gray')
```

Définissez la matrice $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ (`a` comme *array*, tableau) :

```
a = np.array([[1, 0], [0, 0]])
print(a)
```

Affichez l'image correspondant à la matrice (dans une console IPython active) :

```
plt.imshow(a, interpolation='none')
plt.show()
```

`interpolation='none'` est là pour éviter que Python ne lisse le bord des pixels avec un dégradé.
On constate que les chiffres de la matrice `a` sont interprété comme de l'intensité lumineuse :

- la valeur minimale (ici, 0) : noir
- la valeur maximale (ici, 1) : blanc

Changer la valeur d'une case (appelée *pixel*) pour obtenir du gris. Rappel de syntaxe : `a[i,j] = ...`.

B) Manipulation de tableaux Numpy

1) Masques

En console, après avoir exécuté une fois votre programme de départ (et donc chargé Numpy, Matplotlib, etc.) via l'option « Exécuter dans l'interpreteur Python ou IPython courant » du menu accessible via F6, tester les commandes suivantes (et les comprendre !) :

1)

```
>>> a > 0.5
>>> a[a > 0.5] = .7
```

2) Tirer un tableau 20×20 au hasard, et affichez le :

```
a = np.random.rand(20, 20)
plt.imshow(a, interpolation='none')
plt.show()
```

Mettre toutes les valeurs de plus de 0.5 à 1, et celle de moins de 0.5 à 0.

Quelle opération semblez-vous avoir fait sur l'image de départ ?

2) Découpage en tranches

Je vous rappelle ce que dit la documentation sur le sujet :
`https://docs.python.org/3.5/library/stdtypes.html#common-sequence-operations`

`s[i:j]`	slice of `s` from `i` to `j`
`s[i:j:k]`	slice of `s` from `i` to `j` with step `k`

À partir d'un tableau de zéros (`a = np.zeros((20, 20))`), affichez des rayures, des carrés, faites des boucles. Pour récupérer un bloc : `a[i:i+k, j:j+l]`.

Pour sauver vos oeuvres (argument : nom du fichier au format `string` et la matrice des pixels) :

```
plt.imsave('MaBelleImage.jpg',a)
```

C) Charger une image

Chargez l'image « js.jpg » préalablement sauvée dans le répertoire courant (celui où est sauvé votre script python) :

```
a = plt.imread('js.jpg')
```

1) L'afficher.

2) Déterminer la taille (nombre de lignes, nombre de colonne) de l'image.

3) Afficher, avec print, un bloc de taille raisonnable. Quel est le type des données ? Avec `a.max()` et `a.min()` déterminer les valeurs minimales et maximales.
Pour information : l'intensité lumineuse est souvent stockée sur un octet (= 8 bit), donc entre 0 et $2^8 - 1 = 255$.

D) Exercices

On testera les différentes fonctions sur l'image précédente.

Exercice 1
Écrire une fonction qui crée le négatif d'une image.

Exercice 2 (flou)
Le flou le plus classique est le « flou gaussien » : on utilise la gaussienne ($x \mapsto e^{-x^2}$, comme dans le DL) que l'on convole ($f \star g(x) = \int_a^b f(t)g(x-t)\,\mathrm{d}t$, sauf qu'ici on somme sur tous les pixels) avec l'image. Nous allons faire plus basique.

1) À l'aide d'une moyenne raisonnable sur les blocs de 9 pixels, proposer un algorithme pour rendre floue une image.
Indication : *Ne pas se compliquer la vie : ignorez les bords.*

2) Pour que l'effet soit visible, il faut prendre des blocs plus importants : adaptez votre fonction pour que les blocs soient de taille paramétrable.

Exercice 3 (baisse de résolution)
Sur le même modèle que l'exercice précédent, proposer un algorithme de baisse de la résolution d'une image (on pourra diviser par 4 le nombre de pixels, ou par N^2 pour les plus audacieux). Indication : *Ne pas se compliquer la vie : considérez que votre image a un nombre de ligne et de colonne divisible par N. Si ce n'est pas le cas pour votre image test, extrayez une sous-image de taille ad hoc.*

Exercice 4 (augmentation du contraste)
On va appliquer une fonction « en S » par rapport à $y = x$: on cherche f impaire définie sur $[-1, 1]$ telle que

$$f(0) = 0 \qquad f(0.5) = \alpha \qquad f(1) = 1$$

On construit une courbe affine par morceaux.

1) Écrire une fonction `normalise` qui prend un tableau a, dont les éléments varient entre `a.max()` et `a.min()`, et retourne un tableau de float64 dont les éléments varient entre -1 et 1.

2) Écrire une fonction `fct_contraste` d'arguments `x` et `alpha` qui retourne le résultat $f(x)$ voulu.

3) Nous allons vectoriser cette fonction :

```
vfct_contraste = np.vectorize(fct_contraste)
```

Désormais on peut appliquer `vfct_contraste` à un tableau Numpy sans se poser de questions :

```
b = vfct_contraste(a, .75)
```

II) Images couleurs

Si l'image est en couleur, le tableau contient des triplets d'entiers, chacun représentant l'intensité en rouge (`R`), vert (`G`) et bleu (`B`).

1) Tester avec un tableau aléatoire.

2) Mettre à 0 les composantes verte et bleue dans l'image de départ.

3) Extraire les images (niveau de gris) représentant les composantes rouge, verte et bleue.

4) Charger le fichier `couleur.jpg`

5) Adapter quelques unes des fonctions de la partie I au cas d'une image couleur.

6) Passer l'image en niveaux de gris.

III) Arbre récursif

Après avoir codé une fonction qui permet de tracer un segment (selon la pente du segment la fonction diffère), coder l'arbre récursif du premier TP.

IV) Fractales

(Annales 0 de l'oral II de mathématiques et algorithmique.)
On pose $M = 20$ et $m = 10$. À un nombre x quelconque, on associe la suite (u_n) définie par

$$u_0 = 0 \qquad \text{et} \qquad u_{n+1} = u_n^2 + x \quad \text{pour} \quad n \geqslant 0$$

S'il existe, on note k le plus petit entier tel que l'on ait $0 \leqslant k \leqslant m$ et $|u_k| > M$.
On définit alors la fonction f par

$$f : x \mapsto \begin{cases} k & \text{si } k \text{ existe} \\ m+1 & \text{sinon} \end{cases}$$

1) Coder f.

2) Tracer l'allure de la courbe représentative de la fonction f sur $[-2, 2]$, en créant une liste `LX` de 401 valeurs équiréparties entre -2 et 2 inclus.

Indication : *On pourra utiliser les fonctions `plot` et `show` de la sous-bibliothèque `matplotlib.pyplot`.*

3) Construire le tableau des valeurs $f(x+iy)$ où x prend 101 valeurs comprises entre -2 et 0.5 et y prend 101 valeurs entre -1.1 et 1.1.

On rappelle que le nombre complexe i est représenté par `1j`. Par exemple, le complexe $1 + 2i$ est représenté par `1+2j`.

4) Tracer l'image que code ce tableau. Quels paramètres peut-on modifier pour obtenir une meilleure résolution ?