TP 5 correction
k moyennes

L'algorithme des k-moyennes peut être utilisé pour réduire le nombre de couleurs (en) d'une image sans que cela ne nuise trop à sa qualité. L'espace de couleur du système RGB de codage des couleurs consiste à représenter une couleur par trois nombres entiers compris entre 0 et 255 qui représentent les intensités respectives du rouge, du vert et du bleu dans la couleur à afficher. Cela donne 2563 couleurs possibles, soit environ 16 millions. L'œil humain non-entraîné peine à distinguer autant de couleurs et il est donc possible de remplacer deux couleurs proches par une seule sans grande perte de qualité ; ce peut être utile à des fins de compression ou pour permettre l'affichage optimal d'une image sur un écran ou une imprimante n'offrant pas une grande variété de couleurs. L'algorithme des k-moyennes permet de trouver, pour chaque pixel d'une image, parmi une liste de k couleurs définies, la couleur qui est la plus proche.

L'algorithme de réduction d'image nécessite plusieurs itérations. D'abord on initialise en choisissant une liste de k couleurs, et on crée une nouvelle image en remplaçant chaque pixel de l'image d'origine par la couleur dont il est le plus proche. À chaque itération, ensuite, on récupère tous les pixels d'une même couleur pour créer une partition. Pour chacune des k partitions obtenues, on calcule la moyenne des couleurs, ce qui donne une nouvelle liste de k couleurs. On remplace alors chaque pixel de l'image d'origine par la couleur dont il est le plus proche, ce permet d'obtenir une nouvelle image.

Le partitionnement des pixels puis le remplacement de leur couleur par une couleur moyenne est itéré jusqu'à ce que l'image ne soit plus modifiée par le procédé.

Les paramètres qui influent sur le résultat sont :

— la valeur de k ;
— le choix d'initialisation des couleurs ;
— l'arrêt éventuel de l'exécution avant que les couleurs ne soient complètement stabilisées.

1 Importation d'une image

On peut représenter une image (noire et blanc ou couleur) grâce à une matrice ou chaque élément a_{ij} représente une case ou un pixel de cordonnée (i, j) qui est de couleur la valeur a_{ij} qui est un 3-uplet (r, bg) d'entiers entre 0 et 255, (rouge, vert bleu). On importera les modules, `numpy`, `numpy. random` et `PIL` pour traiter les images.

```
>>>import numpy as np
>>> import numpy.random as rd
>>> import PIL.Image
>>> photo=PIL.Image.open('chemin.photo.png') # ouvre le fichier photo sous Pythor
# il faut rentrer l'adresse du chemin par exemple:
>>> photo=PIL.Image.open('C:\\Users\\eleve\\Documents\\Python\\photo.png')
>>> photo.show() # affiche l'image
>>> photo.size # indique la taille
(201,251)
```

Il faut, pour travailler sur une photo, d'abord la transformer en tableau (matrice). On utilise, pour cela, la bibliothèque `numpy` :

```
>>> import numpy as np
>>> phototab=np.array(photo) # Transforme la photo en tableau
>>> photo2=Image.fromarray(phototab) # Transforme un tableau en image
>>> photo2.save('photo.jpg') # Permet de sauvegarder une image sous un format au
```

Pour rechercher ou déplacer des fichier avec Python : Les commandes qui suivent doivent être lancées après `import os`.

— savoir quel est le répertoire courant : `os.getcwd()`
— connaître le contenu du répertoire courant : `os.system('truc')` où `truc` est la commande système de l'O.S. [1] sous lequel **Python** est exécuté [2].
— changer de répertoire courant : `os.chdir('chemin_du_repertoire')`

On importera l'image, qu'on stockera dans `photo` , puis qu'on transformera en tableau dans la variable `phototab`.

2 Compression d'image via k_moyenne

À l'initialisation, une palette de k couleurs est choisie aléatoirement et chaque pixel est rattaché à la couleur de la palette de laquelle il est le plus proche. Lors d'une itération, on remplace chaque couleur de la palette par la «moyenne» des couleurs des pixels qui y sont rattachés. On rattache à nouveau chaque pixel à la couleur la plus proche dans la palette. On poursuit jusqu'à la convergence, c'est-à-dire jusqu'à ce qu'une itération ne modifie plus les couleurs de la palette.

1. Programmer la fonction `dist(x,y)` prenant comme paramètre deux 3-uplet d'entier, correspondant à deux couleurs, et renvoyant

```
import numpy.random as rd
import numpy as np
def dist(x,y):
     return np.sqrt(np.vdot(x-y,x-y))

def barycentre(X,xi):
   """ X ensemble de points, xi liste d'indices de X.
   Renvoie le barycentre des X[k] pour k dans xi """
   s=np.array([0,0,0])
   for k in xi:
        i,j=k
        s=s+X[i,j]

   return np.floor(s/len(xi))# ou aussi s//len(xi)
def initialisation(n,p,k):
   """ initialisation de l'algorithme : renvoie une partition de [n*p]
   en k morceaux non vides."""
   assert n*p>k
   part=[[] for i in range(k)]
   I=rd.randint(0,k,(n,p))
   for j in range(n):
       for i in range(p):
        part[I[j,i]].append((j,i))
   return part
def Newphoto(X,part):
    """ M la liste des couleurs des barycentres, X la photo,
     part la liste des couples des points repartie"""
    n,p,c=np.shape(X)

    k=len(part)
    M=[barycentre(X,xi) for xi in part]
    Nphoto=X.copy()
```

1. *Operating System* ou système d'exploitation
2. ainsi, toutes les commandes systèmes peuvent être effectuées à partir de **Python**, effacement de fichiers, de répertoires, changement de répertoire, etc. : DANGER!!!!

```
    for s in range(k):
        for x in part[s]:
            i,j=x
            Nphoto[i,j]=M[s]

    return Nphoto

def plus_proche(p,M):
    i, d = 0, dist(p,M[0])
    for j in range(1,len(M)):
          if dist(p,M[j])<d:
                i, d = j, dist(p,M[j])
    return i
def k_moyennes(X,k):
   n,p,c=np.shape(X)
   assert n>=k
   part=initialisation(n,p,k)
   M = [barycentre(X,xi) for xi in part]

   while True:

            part2 = [[] for _ in range(k)]
            test=0
            # for i in range(n):
            #     for j in range(p):
            #         s=plus_proche(X[i,j],M)
            #         part2[s].append((i,j))
            #         if test==0:
            #             if (i,j)not in part[s]:
            #                 test=1
            for u in range(k):
                for x in part[u]:
                    s=plus_proche(X[x],M)
                    part2[s].append(x)
                    if s!=u:
                       test=1


            M = [barycentre(X,xi) for xi in part]

            if test == 0:
                    return part2
            else:
                  part=part2.copy()
```

3 Modification et test qualité

Modifier la fonction `dist()` pour que les bleus soit mis en valeur. Faire des tests.
Programmer la fonction risque, représentation graphique en fonction de k.