Premiers pas dans l'utilisation de `matplotlib`

J'utilise l'exercice 6 de la feuille no 3 comme point de départ de cette discussion.

Introduction (les points 1) et 2) de l'exercice)

On importe la figure `cBird.jpg` en utilisant la commande `plt.imread`.

Figure 1: L'image `cBird.jpg` et sa décomposition (en trois matrices, `im_R`, ...) en utilisant les trois couleurs primaires

Avec les commandes

```
im_array = plt.imread("cBird.jpg")
print(np.shape(im_array))

im_R = im_array.copy()
im_R[:, :, 1:] = 0
```

```
6  im_G = im_array.copy()
   im_G[:, :, 0] = 0
8  im_G[:, :, 2] = 0
   im_B = im_array.copy()
10 im_B[:, :, :2] = 0

12 plt.imshow(im_R)  # la composante des rouges
```

on voit que `im_array` est un tableau de taille $512 \times 512 \times 3$. C'est un tableau formé de trois matrices 512×512, chacune contrôlant une des couleurs primaires de l'image. Par exemple, dans le code ci-dessus, à la ligne 5, on extrait la composante "rouge" de l'image — dans la figure 1 on a représenté l'image initiale ainsi que les trois images des couleurs primaires qui la compose.

Remarque. Chaque pixel a trois composantes, trois entiers R, G et B compris entre 0 et $255 = 2^8 - 1$.

Autour du point 6) de l'exercice

Essayer des opérations vectorielles sur `im_array` *et afficher les résultats sous forme d'images. Par exemple remplacer les composantes de chaque couleur par leur moyenne.*

On conçoit le tableau `im_array` comme étant composé de trois matrices (de trois feuilles), une pour chaque couleur primaire. Dans cette question, on veut faire la moyenne dans cette direction, transverse aux feuilles. Un exemple de fonction répondant à cette question est le suivant. Voir la figure 2, l'image de gauche.

```
def nuancesDeGris(x_array):
2    """
     x_array est un tableau de trois dimensions (H, L, 3).
4    On cree une copie de x_array et on remplace chacune
     des trois matrices (H, L) par la  moyenne des trois.
6    On retourne ce nouveau tableau.
     """
8    res = x_array.copy()
     moyenne = np.sum(x_array, axis=2)//3
10   for k in range(3):
         res[:, :, k] = moyenne
12   return res
```

Couleurs et angles. On peut transformer en nuances de gris toutes les couleurs sauf celles proche d'**orange**, par exemple. La couleur orange est défini par le triplet $[255, 128, 0]$. On voudrait conserver les pixels qui ont la couleur proche de ce triplet.

Pour décider si deux couleurs sont proches, on utilise la représentation de la couleur *hsv*, c'est-à-dire *hue* (angle), *saturation*, *value* (valeur). La fonction suivante retourne l'angle de la couleur `RGB`.

```
def angleDeLaCouleur(RGB):
2    """
     RGB est un triplet (liste ou vecteur) d'entiers
4    representant une couleur.  On retourne l'angle de la
```

Figure 2: En nuance de gris et en nuance de gris tout en conservant les oranges

```
    couleur calcule avec la fonction rgb_to_hsv.

    Cette fonciton travaille seulement avec des triplets entre
    0 et 1.
    """
    if type(RGB==list):
        RGB = np.array(RGB)
    rgb = RGB/255
    hsv = rgb_to_hsv(rgb)
    return hsv[0]*360
```

L'image en nuance de gris avec les couleurs ayant l'angle proche de celui d'un angle donné est obtenue par exemple avec la fonction suivante.

```
def nuancesDeGrisSauf(x_array, RGB_test, dangle=15.):
    """
    x_array est un tableau de trois dimensions (H, L, 3).
    On cree une copie de x_array et on remplace chacune
    des trois matrices (H, L) par la  moyenne des trois la ou
    l'angle de la couleur du pixel est plus loin que dangle
    par rapport a l'angle de RGB_test.
    On retourne ce nouveau tableau.
    """
    H, L, C = np.shape(x_array)
    angle_test = angleDeLaCouleur(RGB_test)
    res = x_array.copy()
    moyenne = np.sum(x_array, axis=2)//3
    for i in range(H):
        for j in range(L):
            RGB = x_array[i, j, :]
            angle = angleDeLaCouleur(RGB)
            if abs(angle - angle_test) > dangle:
                for k in range(3):
                    res[i, j, k] = moyenne[i, j]
```

```
    return res
```

Cette fonction est très lente ; on devrait la réécrire avec des calculs `numpy`, c'est-à-dire exécutés sur tous les éléments d'un tableau à la fois (voir l'aide de la fonction `rgb_to_hsv`).

Autour du point 4) de l'exercice

Zoomer sur une partie de l'image en choisissant une partie de ***`im_array`****.*

Zoomer dans l'image signifie l'affichage d'un sous-tableau seulement. On utilisera les sous-tableau par la suite pour rendre l'image floue.

```
def sous_tableau(tableau, NW_pixel, longeur):
    """
    tableau est un tableau avec trois dimensions.
    NW_pixel est une liste [i, j] qui represente le pixel en haut
    a gauche du sous-tableau voulu.
    longeur est un entier positif representant la longeur dans les
    deux premiere dimensions du sous-tableau voulu.

    Retourne le sous-tableau.
    """
    i0, j0 = NW_pixel
    i1 = i0 + longeur
    j1 = j0 + longeur
    return tableau[i0:i1, j0:j1, :]

def pour_zoom(tableau, C_pixel, longeur):
    """
    tableau est un tableau-image avec trois dimensions.
    C_pixel est une liste [i, j] qui represente le pixel au centre
    du sous-tableau qui sera retourne.
    longeur est un entier positif representant la longeur dans les
    deux premiere dimensions du sous-tableau retourne.

    Retourne le sous-tableau qui peut etre construit a l'interieur
    du tableau en partant de C_pixel et longeur.  Il y a une erreur
    si le depassement se fait vers le bas ou la droite.
    """
    if longeur%2==0:
        longeur += 1

    i, j = C_pixel
    i = i - (longeur - 1)//2
    j = j - (longeur - 1)//2
    NW_pixel = [int(max(i, 0)), int(max(j, 0))]
    return sous_tableau(tableau, NW_pixel, longeur)
```

Rendre l'image floue. Une fonction pour rendre l'image floue est un peu plus compliquée à écrire si on veut agir sur tous les pixels de l'image. Le principe est de remplacer chaque pixel par la moyenne des pixels voisins (dans les deux premières directions). On veut utiliser la puissance de `numpy` pour faire des calculs sur toutes les composantes des tableaux simultanément. Pour cette raison, on laissera une bordure de pixels non changés par rapport à l'image de départ.

Figure 3: Image floutée avec $d = 5$

```python
def pour_rendreFlou(tableau, d):
    """
    2d + 1 est le cote du rectangle des pixels qui sont utilises
    pour calculer la moyenne qui remplace le pixel du milieu dans
    le tableau-image retourne.
    """
    H, L, C = np.shape(tableau)
    res = tableau.copy()
    h = H - 2*d
    l = L - 2*d
    c = C
    S = np.zeros((h, l, c), dtype=int)
    for i in range(0, 2*d+1):
        for j in range(0, 2*d+1):
            S += sous_tableau(tableau, [i, j], H - 2*d)
    S = S//((2*d + 1)**2)
    res[d: d+h, d: d+l, :] = S
    return res
```

Fichiers `python`

Les fichiers contenant les codes ci-dessus sont les fichiers suivants.

- `f3_images.py`
- `f3_images_2.py`

Dans le fichiers `f3_couleurs.py` on construit un gradient circulaire basé sur la manipulation des couleurs sous la forme `hsv`.