L'objectif du T.P. qui suit (dont l'autrice initiale est Mme Durand, professeure au lycée Déodat de Séverac) est de manipuler des images avec les modules `numpy` et `matplotlib`. L'ensemble des fichiers image à utiliser, ainsi que le texte de ce TP - en couleurs ! - sont à récupérer sur le site de la classe et à enregistrer dans votre répertoire personnel.

1 - Prise en main

Les images que l'on va manipuler sont des images dites matricielles. Elles sont composées d'une matrice (tableau ou `numpy.ndarray`) de points colorés appelés pixels. On utilisera en particulier le format PNG en « couleurs vraies ». Dans ce format, les couleurs sont codées selon le mode RVB (rouge, vert, bleu) ou RGB en anglais. Pour chaque pixel, la valeur de chacune des couleurs primaires correspond à un nombre réel variant entre 0 (absence) et 1 (intensité maximale). Comme il s'agit d'une synthèse additive des couleurs, le noir sera codé par [0,0,0], le blanc par [1,1,1] et un pixel « pur rouge » sera codé par [1, 0, 0].
Nous allons aborder, ici, une manipulation de « bas niveau » de l'image en la traitant pixel par pixel à l'aide d'un code python. Nous utiliserons pour cela la bibliothèque `matplotlib.image` pour ouvrir des images existantes et la bibliothèque `matplotlib.pyplot` pour afficher les images.

1.1 - Avant de commencer

— Ouvrir la distribution Python de votre choix (Pyzo, Spyder,...) puis déterminer le répertoire de travail en tapant les instructions suivantes dans le shell :

```
>>> import os
>>> os.getcwd() # get current working directory
```

— Télécharger, sans l'ouvrir, l'image `portrait.png` depuis le site *Cahier de prépa* de la classe puis l'enregistrer dans le répertoire de travail de Python (que l'on ne va pas changer).

1.2 - Importer une image

Après avoir importé la bibliothèque `matplotlib.image` à l'aide de l'alias `img`, on peut importer une image de format `.png` à l'aide de l'instruction `img.imread(adresse de l'image)`.
Taper dans un *éditeur*, les commandes suivantes puis les exécuter (touche F5) :

```
import matplotlib.image as img
import numpy as np
ada = img.imread ("portrait.png")
print(ada)
print(type(ada))
print(len(ada))
print(len(ada[0]))
print(len(ada[0,0]))
print(np.shape(ada))
```

Question 1 Expliciter sous quelle forme (type d'objet, structure,...) l'image est importée, c'est-à-dire le contenu de la variable `ada`.
Question 2 Quelles autres informations avez-vous fait afficher ?

1.3 - Afficher une image

Pour afficher une image quelconque importée dans la variable `image`, on utilise `plt.imshow(image)` après avoir importé la bibliothèque `matplotlib.pyplot` à l'aide de l'alias `plt`.

Taper à présent, dans l'éditeur, les deux instructions suivantes :

```
import matplotlib.pyplot as plt
plt.figure(1)
plt.imshow(ada)
plt.show()
```

Question 3 Que représentent les graduations sur l'image ?
Question 4 Savez-vous de qui est-ce le portrait ?

1.4 - Conversion en niveau de gris

Image au format RVB
Dans une image en niveaux de gris au format RVB, les trois composantes R, V, B de chaque pixel ont la même valeur. Celle-ci vaut 0 pour un pixel noir, 1 pour un pixel blanc. Pour convertir une image couleur en une image en niveaux de gris, il faut remplacer les trois intensités de chaque couleur primaire en une seule intensité de gris. Une première idée pourrait être de faire la moyenne de ces trois intensités (en effet, plus une couleur est intense, plus l'image est globalement foncée). Ceci dit, l'œil est plus sensible à certaines couleurs qu'à d'autres. Le vert (pur), par exemple, paraît plus clair que le bleu (pur). Pour tenir compte de cette sensibilité dans la transformation d'une image couleur en une image en niveaux de gris, on ne prend généralement pas la moyenne arithmétique des intensités des trois couleurs fondamentales, mais une moyenne pondérée. La formule standard donnant le niveau de gris en fonction des trois composantes est :

$$gris = 0.299 \times rouge + 0.587 \times vert + 0.114 \times bleu$$

Question 5 Écrire une fonction `niveaux_gris(image)` prenant en argument un tableau numpy `image` provenant de l'importation d'une image couleur quelconque et retournant le tableau numpy correspondant à cette image en niveaux de gris. Les trois niveaux R, V, B d'un pixel sont égaux au niveau de gris donné par la formule ci-dessus. Le résultat attendu est donné figure 1.
Notons que l'on définit une tableau numpy de dimension `(n,p,q)` ne contenant que des zéros à l'aide de l'instruction `np.zeros((n,p,q))`.

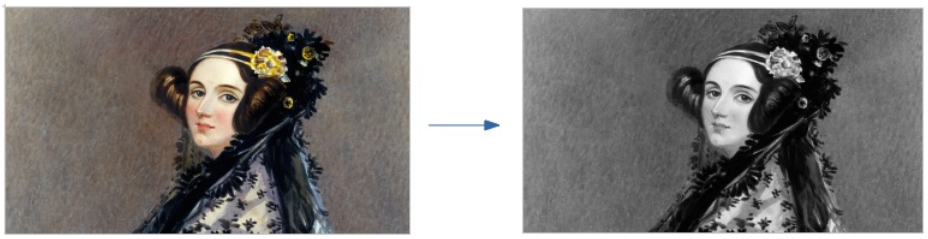

FIGURE 1 – Conversion d'une image couleur en niveaux de gris.

Image au format B
Dans une image en niveaux de gris, les niveaux de rouge, vert et bleu étant identiques, les informations sont redondantes. Il existe un format spécial plus adapté pour stocker des images en niveaux de gris (inutile de répéter trois fois la même valeur !) : le format B. Il est possible de sauvegarder la matrice des pixels non pas en associant à chaque pixel un triplet de niveau (R,V,B) mais une valeur unique égale au niveau de gris. (Le « B » du nom de format signifie simplement « black ».)
Question 6 Quelle instruction suffit-il d'écrire pour transformer une image en niveaux de gris au format RVB en une image en niveaux de gris au format B ? (une ligne !)
Question 7 Visualiser l'image au format B ainsi obtenue. Qu'observe-t-on ?
Pour remédier à ce problème, on peut ajouter `"gray"` dans les arguments de la fonction `imshow` :

```
plt.imshow(image, "gray")
```

2 - Analyse d'un cliché de nanoparticules

Les nanoparticules métalliques sont l'objet de nombreuses recherches et de nombreuses applications depuis plusieurs dizaines d'années. Leurs propriétés sont largement liées à leur taille. Par exemple, lorsqu'elles sont en suspension, des nanoparticules d'or auront une couleur dépendant directement de leur taille. Ainsi, à l'issue d'une synthèse de nanoparticules, il est très important de pouvoir déterminer la taille des particules fabriquées, et même plus précisément la distribution des tailles (le « graâl » étant d'obtenir un échantillon avec des particules toutes de la même taille !). Nous étudierons dans cette partie un cliché de nanoparticules d'alliage fer-platine obtenu par microscopie électronique en transmission (figure 2).

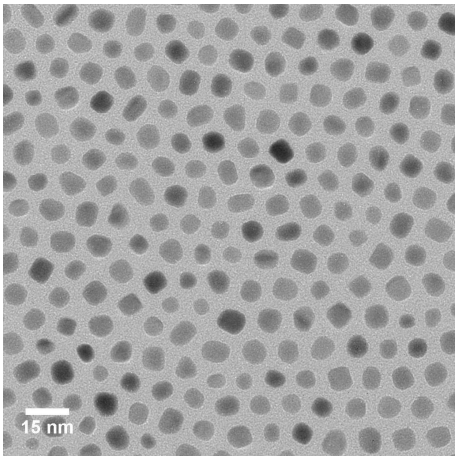

FIGURE 2 – Nanoparticules d'alliage Fe-Pt, cliché obtenu par microscopie électronique en transmission. (Source : Michaël Delalande, Synthèse chimique, structure et propriétés magnétiques de nanoparticules d'alliage Fe-Pt, sous la direction de Yves Samson, Grenoble, Université Joseph Fourier, 2007)

Le but de cette partie est d'analyser l'image afin de déterminer le nombre de nanoparticules présentes.

2.1 - Seuillage de l'image

Pour commencer, il faut effectuer un seuillage de l'image, c'est-à-dire construire une nouvelle image dans laquelle les particules apparaîtront complètement noires (intensité : 0) sur fond complètement blanc (intensité : 1). Autrement dit, il s'agit d'une augmentation du contraste de la façon la plus extrême possible.

Question 8 Écrire une fonction `seuil(imageB, s)` qui effectue l'opération décrite ci-dessus. Elle prend comme arguments une image en niveaux de gris en format B et une valeur seuil `s`, réel compris entre 0 et 1 puis retourne l'image « seuillée ». Tous les pixels ayant une intensité de gris supérieure à `s` deviendront blancs (intensité 1) et tous les pixels ayant une intensité inférieure deviendront noirs (intensité 0).

Question 9 Tester plusieurs valeurs de seuil pour afficher l'image obtenue par seuillage de `Particules.png`, fichier à récupérer sur le site de la classe. Quelle valeur vous paraît la plus adaptée ? Que peut-on dire de la qualité du seuillage obtenu ? Comment pourrait-on y remédier ?

Question 10 Recommencer l'opération de seuillage sur le cliché `Particules_2.png`. Commenter.

2.2 - Reconnaissance des particules

À partir de l'image seuillée précédente, on utilise un algorithme qui permet de reconnaître les différents objets (ici des nanoparticules) présents dans l'image, c'est-à-dire qu'il permet de repérer les différentes composantes connexes présentes dans l'image, en utilisant un système d'étiquetage.
Cette algorithme est expliqué à l'adresse suivante : `www.youtube.com/watch?v=ticZclUYy88`.
Avant de se plonger dans le programme complet permettant d'effectuer cette reconnaissance, il peut être judicieux d'écrire une fonction intermédiaire permettant de déterminer les pixels voisins d'un pixel donné.

Question 11 Écrire une fonction `voisins(tableau, k, l)`, qui prend en entrée un tableau de nombres ainsi que deux indices entiers `k` et `l` et qui retourne deux nombres : les valeurs contenues à gauche et en haut de l'élément situé ligne `k`, colonne `l`.
Attention au cas particulier des bords : si `k=0` (resp. `l=0`), la valeur en haut (resp. à gauche) sera prise égale à -1.

Question 12 Compléter le code de la fonction `connexes(imageB_seuil)` donnée ci-dessous. Cette fonction prend en argument une image seuillée en noir et blanc de particules noires (intensité 0) sur fond blanc (intensité 1) et « cherche » les composantes connexes.
Elle renvoie une image (tableau de pixels) où, à l'intérieur de chaque composante connexe (*i.e.* particule), chaque pixel porte une étiquette identique (un nombre entier à partir de 1), mais différente de celle des autres composantes connexes. Les pixels du fond portent l'étiquette -1. Elle retourne de plus le tableau des équivalences obtenues.
Remarque : il est conseillé de télécharger le fichier python contenant le texte de cette fonction à compléter (fichier `TP9_eleve.py` sur le site de la classe).

```
def connexes(imageB_seuillée):
    resul = imageB_seuillée * -1
    n, p = np.shape(resul)
    E = np.arange(n*p+1) # tableau d'équivalence des étiquettes
    label = 1 # première étiquette

    for i in range(n):
        for j in range(p):
            if resul[i,j] != -1:
                if voisins(resul, i, j) == (-1,-1):
                    resul[i,j] = label
                    label += 1
                elif min(voisins(resul,i,j)) == -1 and max(voisins(resul,i,j)) != -1:
                    # ligne à compléter
                elif min(voisins(resul,i,j)) != -1  and max(voisins(resul,i,j)) !=
                    -1:
                    # ligne à compléter
                    if min(voisins(resul,i,j)) != max(voisins(resul,i,j)):
                        # ligne à compléter (mettre à jour le tableau des
                            équivalences)
    E_final = E[:label]

    for i in range (len(E_final)):
        if E_final[i] != i:
            indice = E_final[i]
            E_final[i] = E_final[indice]

    final = np.copy(resul)

    # scan 2 : on met à jour toutes les étiquettes
    for i in range(n):
        for j in range(p):
            if final[i,j] != -1:
                final[i,j] = E[int(resul[i,j])]

    return(resul, final, E_final)
```

Question 13 Appliquer `connexes` à une image test qui comprend volontairement peu de particules : `Particules_2_detail.png`, en utilisant le code écrit dans le fichier `TP9_eleve.py`. Lorsque cela fonctionne, appliquer alors `connexes` au cliché de nanoparticules à étudier : `Particules_2.png`.

Question 14 Écrire une fonction `Nb_particules(E)` qui prend en argument une liste d'équivalences `E` obtenue à l'issue de l'application de la fonction `connexes` et qui retourne le nombre de particules présentes sur l'image.