TP D'INFORMATIQUE N°6
Algorithme des k moyennes

1 Implémentation des k moyennes

Dans ce TP, nous ferons le choix de représenter un point de $\mathbb{R}^d$ par un **tableau numpy**, pour simplifier et accélérer le calcul des moyennes. On rappelle que les opérations calculatoires, ou l'application de fonctions `numpy` telles que `np.sqrt`, se font termes à termes sur les tableaux numpy. Dans la suite, on appelle **point** un tableau numpy de flottants.

1. Importer `numpy` sous le nom `np`.
2. Écrire une fonction `distance` prenant en argument deux points ayant le même nombre de coordonnées, et renvoyant la distance euclidienne entre ces points. Cette fonction devra utiliser la spécificité des tableaux numpy, et donc se passer de boucle `for`.
3. Écrire une fonction `moyenne` prenant en argument une liste de points et renvoyant leur moyenne (on pourra utiliser `np.zeros` pour initialiser cette moyenne).
4. Écrire une fonction `indice_plus_proche` prenant en argument un point `p` et une liste de points `M`, et renvoyant un indice `i` correspondant à l'indice du point de `M` le plus proche de `p`.
5. En déduire d'implémenter l'algorithme des k-moyennes, dont on redonne le pseudo-code :

```
k_moyennes(X, k):
    soit M une liste de k moyennes
    tant que M est modifiée:
        soit P une liste de k parties vides
        pour chaque point p de X:
            soit i l'indice de la moyenne la plus proche de p
            on ajoute p à la partie d'indice i
        on recalcule la moyenne de chaque partie
    on renvoie P et M
```

On choisira d'initialiser `M` en prenant les `k` premiers points de `X`.

On pourra utiliser `np.array_equal` pour tester l'égalité de deux listes contenant des tableaux numpy.

2 Test de k-moyennes

1. Importer la bibliothèque `random` pour les fonctions `randint` et `random`.
2. Écrire une fonction `generer` prenant en argument deux entiers `k` et `n` et un flottant `e`, tirant `k` points centraux uniformément au hasard dans $[0,1]^2$, puis tirant n points selon les règles suivantes :
   - on tire un indice i au hasard entre 0 et $k-1$,
   - on tire un angle θ au hasard entre 0 et π,
   - on tire une distance d au hasard avec l'instruction `d = np.random.normal(0,e,1)[0]` (loi normale centrée en 0 d'écart-type e),
   - on obtient le nouveau point en ajoutant $(d\cos\theta, d\sin\theta)$ au point central d'indice i.
3. Générer un ensemble de points, l'afficher graphiquement, puis afficher graphiquement la répartition obtenue par la fonction `k_moyennes`.

3 Application à la compression d'image

Le principe de la compression considérée est de remplacer l'ensemble de couleurs présentes dans une image par un ensemble beaucoup plus restreint, que l'on pourrait alors coder par de petits entiers. Il s'agit donc d'une **compression avec pertes**, puisqu'on ne peut plus retrouver l'image initiale à partir de l'image compressée. Cependant, si le choix de l'ensemble restreint de couleurs est bien réalisé, on gardera une grande ressemblance entre l'image compressée et l'image initiale. Nous allons utiliser la fonction `k_moyennes` pour obtenir cet ensemble restreint de couleurs.

1. Importer `matplotlib.pyplot` sous le nom `plt`.
2. Placer l'image `buffon.jpg` dans le répertoire de travail de Pyzo, et l'ouvrir avec l'instruction `img = np.array(plt.imread('buffon.jpg'))`
3. Le pixel de coordonnées i, j d'une image couleur est un triplet d'entiers correspondant aux intensités des couleurs rouge, vert et bleu. Afficher l'image sous Pyzo avec les instructions

   ```
   plt.figure()
   plt.imshow(img)
   plt.title('Image de depart')
   plt.show()
   ```

4. Former une liste `X` de tous les pixels de l'image, et appeler la fonction `k_moyennes` dessus avec $k = 16$ pour obtenir une liste `M` de 16 couleurs moyennes.

   On pourra modifier l'initialisation de `k_moyennes` pour tirer les k moyennes initiales au hasard dans `X`.
5. Définir une image `img2` remplaçant chaque pixel de `img` par la couleur de `M` la plus proche.

   On pourra utiliser l'instruction `img2 = np.zeros((n,p,3),dtype=np.uint8)` pour initialiser `img2`.
6. Afficher `img2`. Les différences avec `img` sont-elles visibles ?