TP 2: algorithmes de clustering
K-means, K-médoïdes, critères de qualité

1- Programmer la méthode des k-Moyennes en vous conformant à l'interface suivante:

```
def my_kmeans(X,K,Visualisation=False,Seuil=0.001,Max_iterations = 100):
```

__X:__ le tableau de taille (N,p) des données à analyser
__K:__ le nombre de clusters à trouver
__Visualisation__ : un drapeau pour visualiser des résultats intermédiaires
__Seuil__ : valeur seuil pour tester la diminution relative de l'erreur quadratique moyenne à chaque itération
__Max_iterations:__ nombre maximum d'itérations

…

```
return C, y, J
```

__C:__ tableau comportant les clusters de dimension (p,K)
__y :__ le tableau (N,1) donnant le numéro du cluster affecté à chaque exemple
__J :__ l'historique de l'erreur quadratique moyenne (MSE) au cours des itérations

Utilisez le squelette de programme my_kmeans.py fourni dans votre ENT

2- Examiner la stabilité de l'algorithme sur plusieurs exécutions

- Pour comparer les exécutions pour une même valeur de K, on s'aidera des critères suivants
  - Nombre d'itérations jusqu'à converge pour un seuil de 0,001
  - Pourcentage de variance expliquée
  - Position des centroïdes

- Effectuer cette analyse pour le dataset IRIS et pour le dataset Breast cancer wisconsin

TP 2: algorithmes de clustering
K-means, K-médoïdes, critères de qualité

3- Améliorer la convergence avec k-means++

Programmer la méthode d'initialisation k-means++ en vous conformant à l'interface fournie et qui est rappelée ici:

```
def initPlusPlus(X,K):
```

X: *le tableau de taille (N,p) des données à analyser*
K: *le nombre de cluster à trouver*

```
return C
```

C: *tableau comportant les clusters de dimension (p,K)*

4- Etudier sur plusieurs exécutions la stabilité de l'algorithme k-means avec l'initialisation k-means++

- Pour comparer les exécutions pour une même valeur de K, on s'aidera des critères suivants
  - Nombre d'itérations jusqu'à converge pour un seuil de 0,001
  - Pourcentage de variance expliquée
  - Position des centroïdes

- Effectuer cette analyse pour le dataset IRIS et pour le dataset Breast cancer wisconsin

... / ...

TP 2: algorithmes de clustering
K-means, K-médoïdes, critères de qualité

5- Programmer la méthode des k-médoïdes

Programmer la méthode d'initialisation k-medoIdes en vous conformant à l'interface fournie et qui est rappelée ici:

```
def my_kmeans(X,K,Visualisation=False,Seuil=0.001,Max_iterations = 100):
```

X: *le tableau de taille (N,p) des données à analyser*
K: *le nombre de clusters à trouver*
Visualisation *: un drapeau pour visualiser des résultats intermédiaires*
Seuil *: valeur seuil pour tester la diminution relative de l'erreur quadratique moyenne à chaque itération*
Max_iterations: *nombre maximum d'itérations*

…

```
return C, y, J
```

C: *tableau comportant les clusters de dimension (p,K)*
y : *le tableau (N,1) donnant le numéro du cluster affecté à chaque exemple*
J : *l'historique de l'erreur quadratique moyenne (MSE) au cours des itérations*

6- Etudier sur plusieurs exécutions la stabilité de l'algorithme des k-medoïdes

- Pour comparer les exécutions pour une même valeur de K, on s'aidera des critères suivants
  - Nombre d'itérations jusqu'à converge pour un seuil de 0,001
  - Pourcentage de variance expliquée
  - Position des centroïdes
- Quel est l'apport de l'initialisation k-means++ sur les k-medoïdes ?
- Effectuer cette analyse pour le dataset IRIS et pour le dataset Breast cancer wisconsin

… / …

TP 2: algorithmes de clustering

7- Comparaison avec scikit learn

Comparer vos méthodes avec les méthodes de scikit learn dans les mêmes conditions d'utilisation

```
################################################
from sklearn.cluster import Kmeans

X = …
# calcul des k-means
kmeans = KMeans(n_clusters=2,init='random',n_init = 10, verbose=1, max_iter=100).fit(X)

#le tableau (N,1) des numéros du cluster affecté à chaque exemple
kmeans.labels_

# les centroïdes résultats
kmeans.cluster_centers
```

```
################################################
from sklearn_extra.cluster import KMedoids import numpy as np

X = …
# calcul des k-medoides
medoids = KMedoids(n_clusters=2, random_state=0).fit(X)

#le tableau (N,1) des numéros du cluster affecté à chaque exemple
kmedoids.labels_

# prediction du medoides le plus proche sur de nouvelles données
kmedoids.predict([[0,0], [4,4]])

# les centroïdes résultats
kmedoids.cluster_centers_

# inertie intra
kmedoids.inertia_
```