TD n° 6 : algorithme des k moyennes

Exercice 1. *Choix des centres initiaux*
Écrire une fonction d'entête

```
def init_centres(datas: [tuple], k: int) -> [int]:
```

où `datas` est une liste de points donnés par leurs coordonnées sous forme de tuple, et qui renvoie une liste de k centres **distincts** (liste de leurs indices dans la liste `datas`) choisis aléatoirement parmi les éléments de `datas`. On pourra utiliser la fonction `randint` du module `random` qui prend deux arguments entiers a et b et qui renvoie un nombre entier compris entre a et b, les deux inclus.

Exercice 2. *Trouver le centre le plus proche*
On suppose avoir écrit une fonction `distance` qui permet de calculer la distance entre deux points donnés par leurs coordonnées (cf TD3 exercice 1).

1. Écrire une fonction d'entête

   ```
   def centre_proche(point: tuple, centres: [tuple]) -> int:
   ```

   qui renvoie l'indice du centre le plus proche de `point` parmi ceux de la liste `centres` (tous les points étant repérés par un tuple de leurs coordonnées). On ne cherchera pas à gérer de manière particulière les cas d'égalité.
2. ★ Modifier la fonction précédente afin que dans le cas où `point` se trouve à équidistance d'au moins deux centres, on choisisse de manière aléatoire parmi ces centres les plus proches.

Exercice 3. *Isobarycentre*
Écrire une fonction `isobarycentre` qui prend en argument une liste de points `groupe`, donnés par leurs coordonnées sous forme de tuple, et qui renvoie l'isobarycentre de cette liste, c'est-à-dire le point (donné par le tuple de ses coordonnées) dont chaque coordonnée est la moyenne de cette coordonnée sur tous les points de la liste `groupe`.

Exercice 4. *Algorithme complet*
Soit une première fonction réalisant la répartition initiale en utilisant les fonctions définies dans les deux premiers exercices :

```
def repartition_init(datas: [tuple], k: int) -> ([int], dict, [int]):
    # Aucune classe donnée au début.
    classes = [-1] * len(datas)
    groupes = {}
    # Initialisation des centres : leurs indices puis leurs coordonées.
    ind_centres = init_centres(datas, k)
    centres = [datas[i] for i in ind_centres]
    # Au départ chacun des k groupes est constitué seulement du centre
    # et les seuls éléments dont on connaît la classe sont les centres.
    numero_groupe = 0
    for i in ind_centres:
        classes[i] = numero_groupe
        groupes[numero_groupe] = [i]
        numero_groupe = numero_groupe + 1
    # Affectation de chaque élément de datas au centre le plus proche.
    for j in range(len(datas)):
        if j not in ind_centres:  # Les centres ont déjà un groupe.
            plus_proche = centre_proche(datas[j], centres)
            groupes[plus_proche].append(j)
            classes[j] = plus_proche
    return ind_centres, groupes, classes
```

1. Quels sont les types et que représentent les trois variables renvoyées ?

On considère maintenant la fonction suivante qui réutilise des fonctions des exercices précédents :

```
1  def k_moyennes(datas: [tuple], k: int, max_iter: int):
2      ind_centres, groupes, classes = repartition_init(datas, k)
3      centres = [datas[i] for i in ind_centres]
4      nb_iter = 0
5      evolution = True
6      while evolution and nb_iter < max_iter:
7          nb_iter = nb_iter + 1
8          if nb_iter >= 2:  # Pas d'évolution à la première étape
9              evolution = False
10         for j in range(len(datas)):
11             plus_proche = centre_proche(datas[j], centres)
12             if plus_proche != classes[j]:
13                 evolution = True
14                 groupes[classes[j]].remove(j)
15                 groupes[plus_proche].append(j)
16                 classes[j] = plus_proche
17         for i in range(k):
18             points = [datas[j] for j in groupes[i]]
19             centres[i] = isobarycentre(points)
20     return centres, groupes
```

2. Quels sont les types et que représentent les éléments de la liste `centres` définie ligne 3.
3. À quelle situation correspondent les lignes 12 à 16 ?
4. En quoi consistent les lignes 17 à 19 ?
5. Expliquer la ligne 6.
6. Quels sont les types des variables renvoyées ?

Exercice 5. *Trouver le minimum global*

1. Écrire une fonction d'entête

   ```
   def variance(datas: [tuple], centres: [tuple], groupes: dict)
                                                          -> float:
   ```

   qui renvoie la variance de la répartition donnée en argument, *i.e.* la somme des carrés des distances entre les points et les centres du groupe auxquels ils appartiennent. La liste `datas` contient les coordonnées des points à répartir, la liste `centres` contient les coordonnées des centres des groupes et `groupes` est un dictionnaire dont les clés sont les numéros des groupes (dans l'ordre des coordonnées de la liste `centres`) et les valeurs les indices des éléments de `datas` dans ce groupe.
2. Compléter les lignes 5 et 8 à 11 de la fonction suivante afin qu'elle renvoie les centres et la répartition en groupes optimale après `nb_repet` répétitions de l'algorithme des `k`-moyennes sur les données `datas`.

```
1  def optimum(datas: [tuple], k: int, nb_repet: int) ->
2                                                  ([tuple], dict):
3      mini = float('inf')  # infini
4      max_iter = 100
5      for
6          centres, groupes = k_moyennes(datas, k, max_iter)
7          var = variance(datas, centres, groupes)
8          if
9
10
11
12     return best_centres, best_groupes
```