INF 413 – TP5-6

Alan Gardin – Ronan Garet

13/03/2017

# Introduction

Le travail effectué consiste en l’étude d’un algorithme de clustering : k-means. Etant donné un ensemble de points, il s’agit de les organiser en « groupes ». Nous utiliserons les notations suivantes :

d : nombre de dimensions

k : nombre de centres

n : nombre de points

# Génération des points

Au début du TP, nous avons généré les points de manière suivant une loi uniforme. Cependant, essayer de définir des centres sur ce genre de données possède assez peu de sens. Notre deuxième approche a donc été de générer des clusters de points de manière aléatoire. C’est ce que fait la fonction *filesManagment.generate\_random\_gaussian\_data*.

Le principe est le suivant : on génère un certain nombre de centres suivant une loi uniforme, puis on associe à chaque point un centre (toujours suivant une loi uniforme). Les coordonnées de chaque point suivent une loi normale de moyenne les coordonnées du centre. L’objectif sera ensuite que les clusters et centre identifiés par l’algorithme soient similaires à ceux de départ.

# Fonctionnement de l’algorithme

L’algorithme prend en entrée une liste de points à coordonnées et , le nombre de barycentres.

L’algorithme renvoie en sortie ou est une liste contenant l’ensemble des points du cluster associé au centre .

1. Choisir les centres initiaux.
2. Calculer les distances des points aux centres
3. Assigner chaque point à un centre.
4. Tant que condition d’arrêt non valide :
5. Calculer les centres des clusters
6. Calculer les distances des points aux centres
7. Assigner chaque point à un centre.

# Calcul des distances :

Dans un 1er temps, nous avons choisi le calcul de distance qui nous a paru le plus naturel : la distance euclidienne. Soit la distance entre les points et , on a donc

# Quantifier la qualité d’une solution

Pour un nombre de centres donnés, une solution est d’autant plus optimale que la distance moyenne entre chaque point et son centre est faible. C’est cette valeur que nous utiliserons pour quantifier la qualité d’une solution par rapport à une autre.

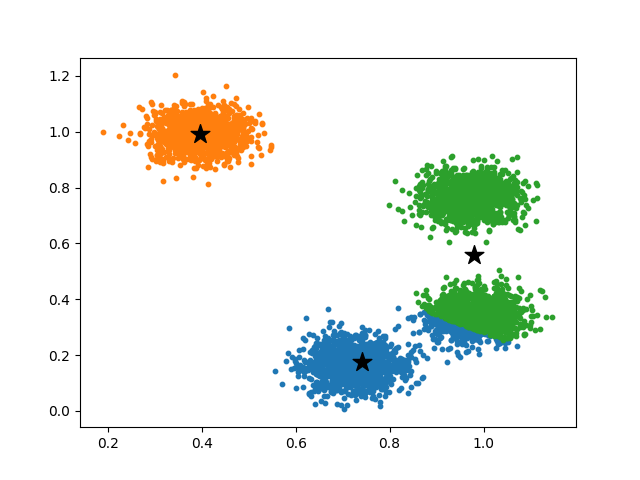
Remarque : Pour comparer des solutions sur des ensembles de points de même taille on se contentera de calculer la somme des distances entre les points et leur centre associé.

# Choix du nombre de centres

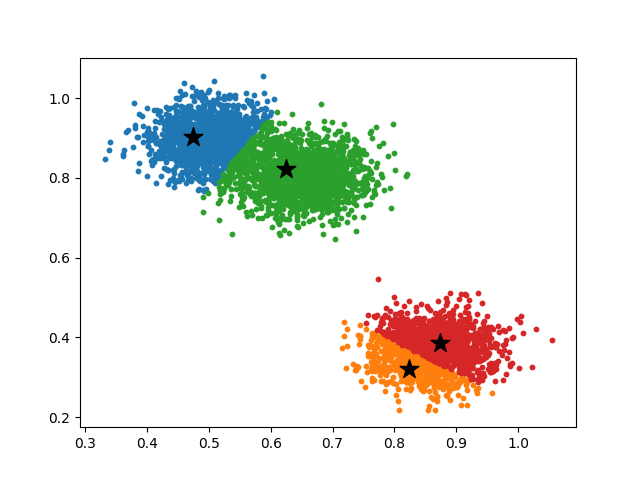
## Algorithme G-means

Dans un premier temps, le nombre de centres sera choisi de manière arbitraire. Ce choix peut être aléatoire, ou alors être suggérée par une connaissance sur la distribution des données.

Mais le choix du nombre de clusters reste un point crucial. En effet, si le nombre de cluster est insuffisant par rapport à la distribution des données, l’algorithme renvoie des résultats « erronés », comme le montre les exemples suivants :



**Exemple d’un nombre de centres insuffisants**



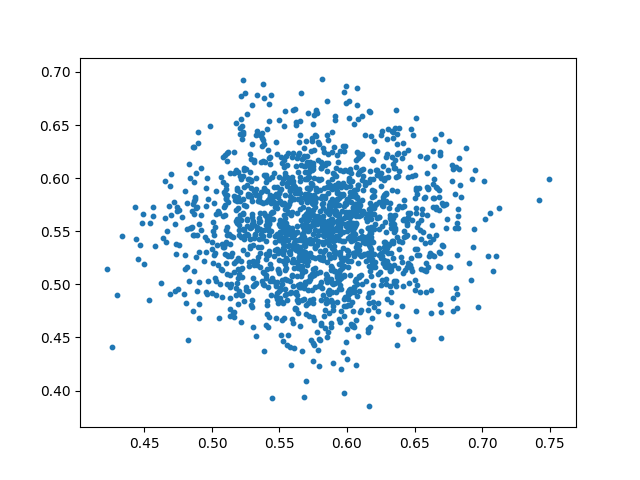
**Exemple d’un nombre de centres trop grand**

Pour pallier ce problème, nous avons implémenté l’algorithme G-means. Celui-ci permet de « trouver » le bon nombre de cluster à adopter. Il prend en entrée un flottant qui décrit à quel « degré » on accepte qu’une distribution donnée soit gausienne. Cet algorithme repose sur le pseudo-code suivant :

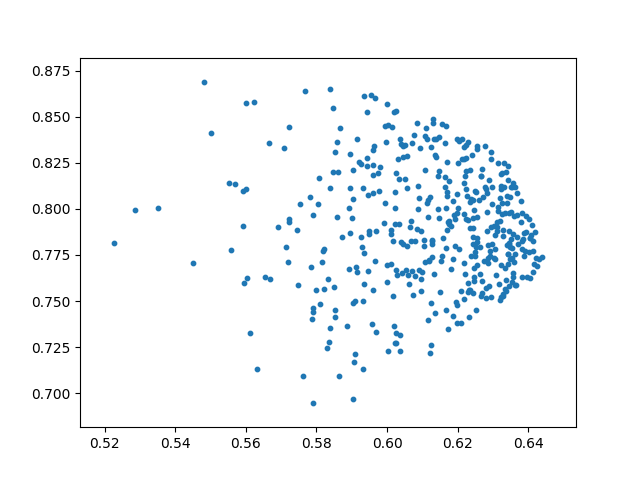
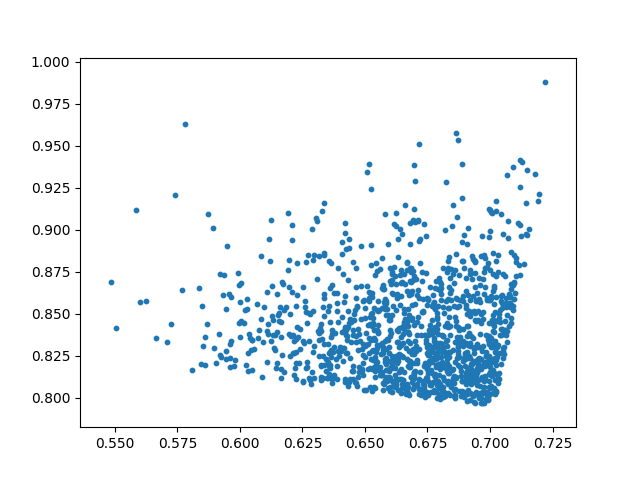
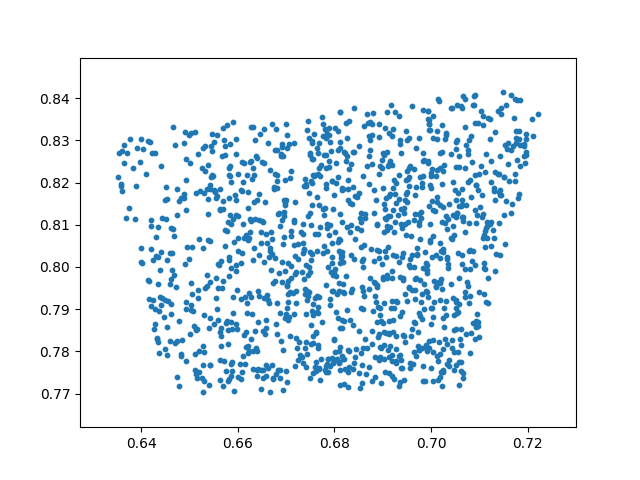
1. k = 1
2. Clusters = k-means(k)
3. Pour chaque cluster dans clusters :
4. Si le cluster non gaussien de paramètre  :
5. k += 1
6. Si un centre a été rajouté, recommencer à l’étape 2.

Dans le pire des cas, cet algorithme exécute fois l’algorithme k-means.

En pratique cependant, la reconnaissance d’une répartition gaussienne s’est avérée assez aléatoire. Par exemple, l’ensemble suivant n’est pas reconnu comme une distribution gaussienne :

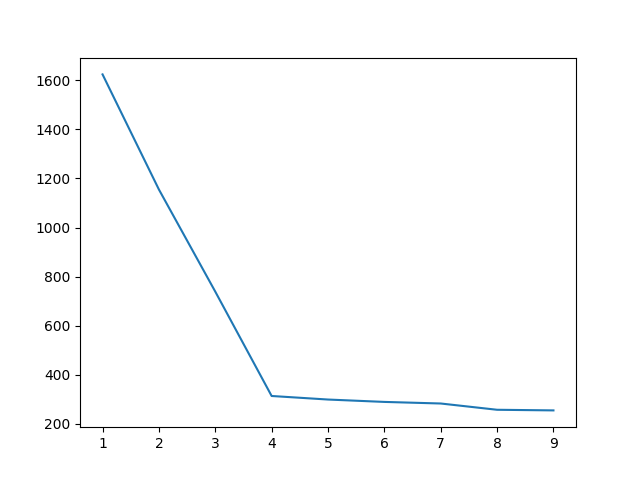
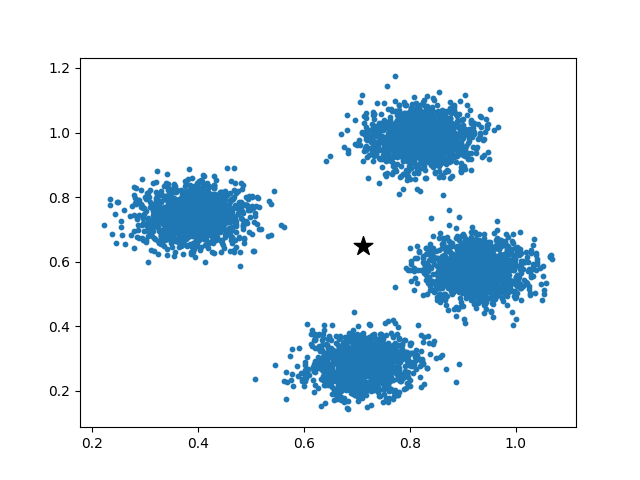


Même s’il ne s’agit vraiment pas d’une distribution gaussienne, cela pose un problème pour la suite de l’algorithme : très souvent l’algorithme ne converge pas : étant donné une distribution gaussienne non reconnue, l’algorithme va couper cette distribution en deux et appliquer à nouveau des tests de répartition gaussienne. Il est facile de voir que la coupe d’une distribution gaussienne a très peu de chance d’être détectée comme gaussienne ensuite. La suite de découpe qui s’ensuit empêche la terminaison de l’algorithme. On obtient par exemple les découpes suivantes :

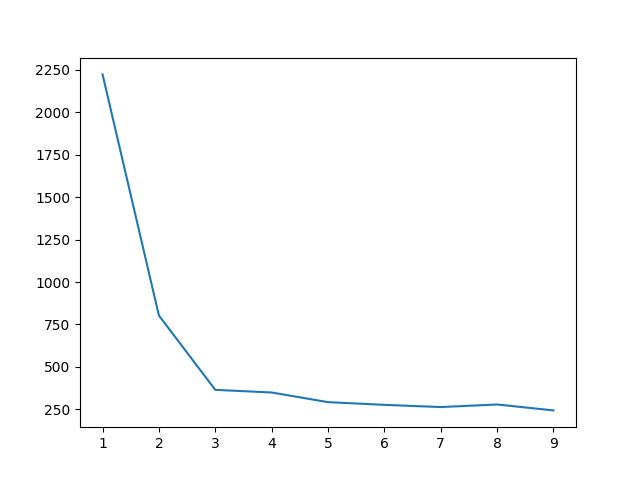
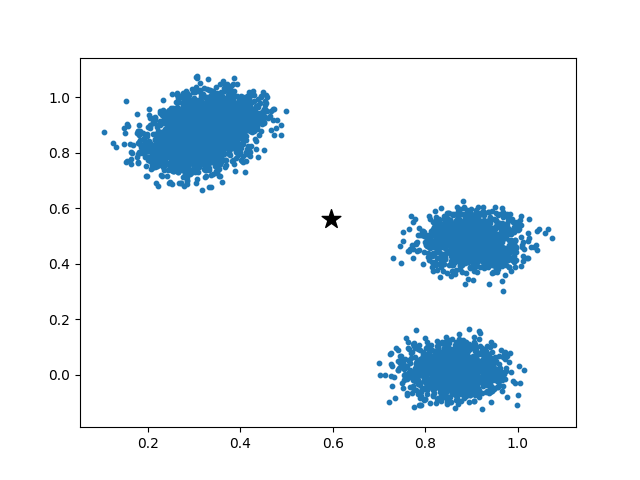


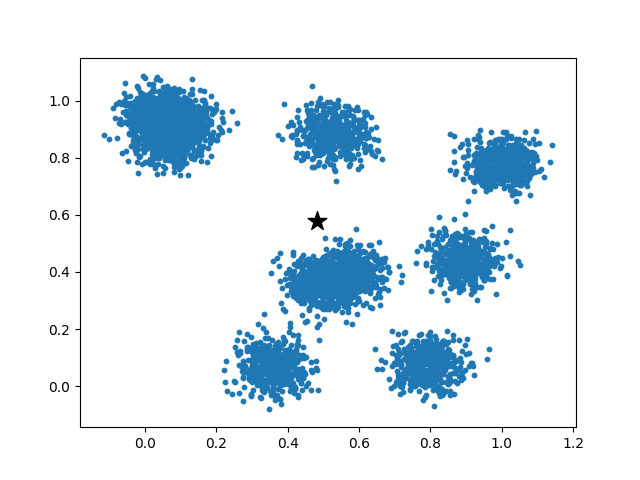
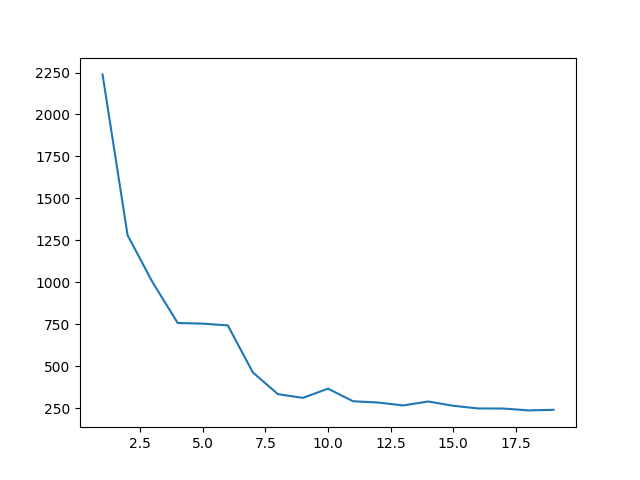
## Une autre approche

Une autre méthode pour calculer le nombre de clusters serait d’afficher la somme des distances entre chaque point et son centre en fonction du nombre de centres choisis. On obtient les résultats suivants.

* 4 clusters, 5000 points

Quand les clusters sont totalement dissociés, on observe un point d’inflexion quand le nombre de centre est égal au nombre de clusters simulés. On remarque également que la 1ère portion de la courbe se rapproche fortement d’une droite.

Les données ci-dessous ont été générées avec les mêmes paramètres que précédemment mais cette fois-ci, deux des clusters se superposent. Du fait du recouvrement, la courbe est moins facile à analyser mais la plus grosse inflexion a toute de même lieu pour 3 centres, ce qui est correct.

* 10 clusters, 5000 points

Pour 10 clusters, la courbe est encore plus complexe. On peut tout de même identifier une zone d’inflexion entre 5 et 8, ce qui permet déjà d’obtenir une idée du nombre de centres.

Nous n’avons pas réussi à implémenter cette méthode mais elle nous semblait tout de même intéressante à présenter.

# Calcul du barycentre

On prend comme barycentre d’un ensemble de n points la moyenne de leurs coordonnées.

# Condition d’arrêt

L’algorithme k-means converge vers une solution localement optimale. L’idéal serait donc d’arrêter le programme quand les solutions trouvées ne varient plus.

Cependant, il est possible que cela n’arrive jamais (présence de cycles).

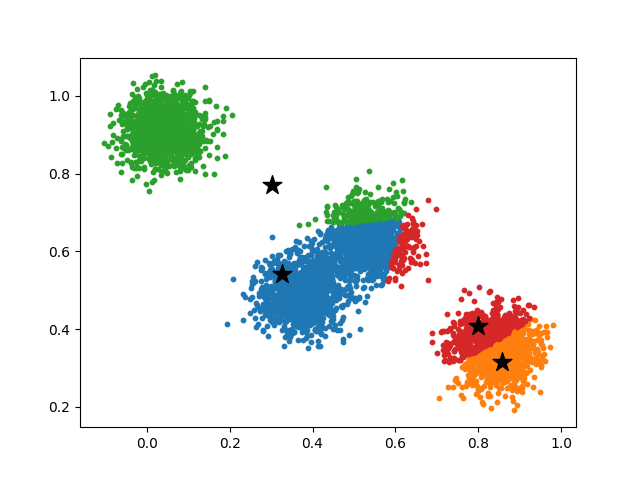
Dans un premier temps, on choisira comme condition d’arrêt un nombre d’itérations choisi de façon arbitraire.

Une autre méthode consiste à garder en mémoire la meilleure solution trouvée (distance moyenne entre les points et leur centre minimale) ; si cette solution n’a pas changé après un certain nombre d’itérations (choisi de façon arbitraire ?) on arrête le programme.

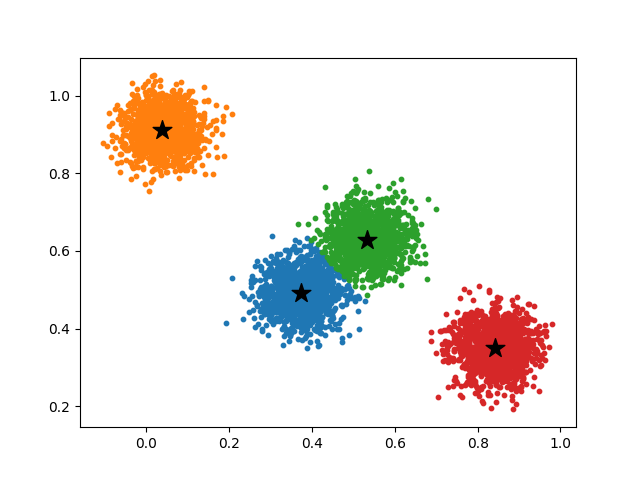
# Choix initial des centres

Dans un premier temps, le choix initial des centres se fera de manière aléatoire. Aucun point n’étant plus important qu’un autre on choisira une probabilité uniforme.

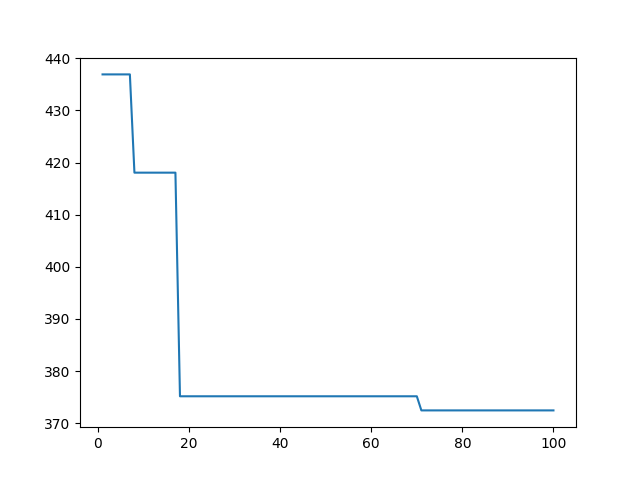
Cependant, l’algorithme des k-means ne fournissant qu’une solution localement optimale, les choix initiaux des centres s’avère crucial.



**Exemple de mauvais résultat dû à un mauvais choix des centres initiaux**

Une façon très simple de résoudre le problème est de lancer le programme plusieurs fois et de garder le meilleur résultat. On augmente ainsi la probabilité d’avoir de « bon » points de départ. Voici le résultat obtenu avec les mêmes points au bout de 50 itérations.

Se pose alors la question du nombre d’itérations. Ce nombre d’itérations dépend bien entendu de la quantité de données et de la puissance de la machine.

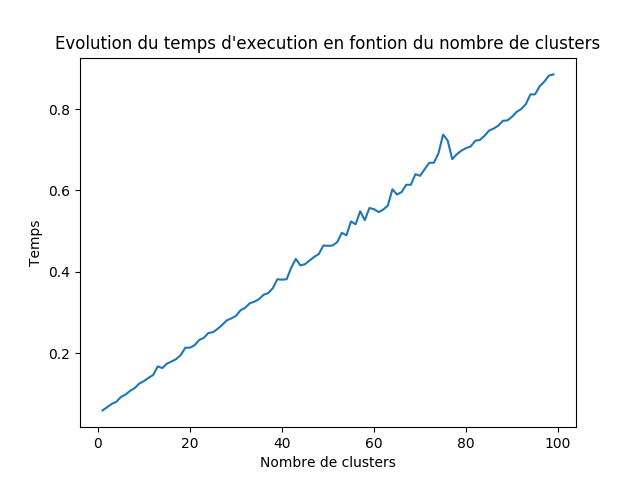
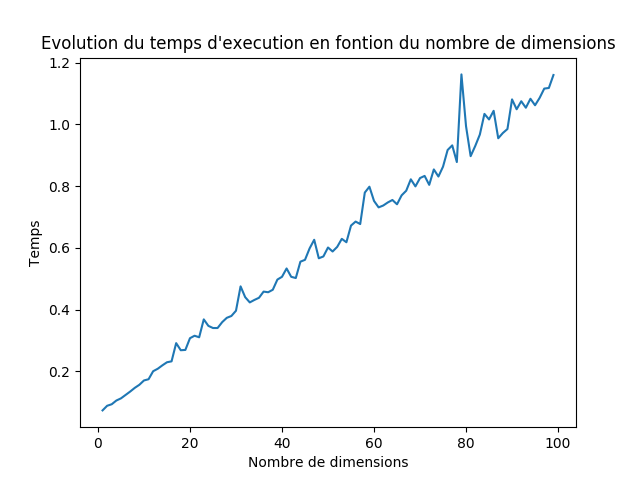
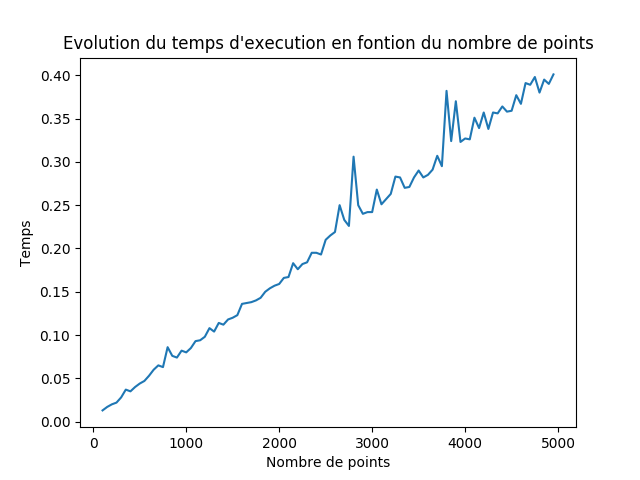
Dans les faits, on observe que quelques dizaines d’itérations suffisent. Voici l’évolution de la meilleure solution en fonction du nombre d’itérations que l’on obtient avec un échantillon de 5000 points répartis sur 10 clusters.

# Complexité de l’algorithme

Le choix des centres initiaux est en .

Le calcul de la distance est en  : pour chaque point, on calcule sa distance avec chaque barycentre, et le calcul de la distance entre points est en .

Le calcul des barycentres est en  : Pour chaque cluster, on fait une addition des coordonnées de ses points qu’on divise par le nombre de points du cluster. Au final, on a un nombre de calculs en

Si on choisit comme condition d’arrêt un nombre donné d’itérations, on a une complexité de l’algorithme en .

Si on se donne comme condition d’arrêt la convergence de la solution, l’algorithme peut ne pas se terminer (possible existence de boucles). On a alors une complexité potentiellement infinie.

# Utilisations possibles de ce genre d’algorithmes

Cet algorithme de clustering peut s’appliquer à de nombreux types de données, pourvu que l’on munisse celles-ci d’une distance. Par exemple, nous avons rendu l’algorithme compatible avec des mots. La distance entre les mots était celle de Levenshtein.

Identification de groupes d’individus dans une foule (une dimension par caractéristique étudiée : âge, revenus, …).

Identification de regroupements d’étoiles afin de déterminer des galaxies