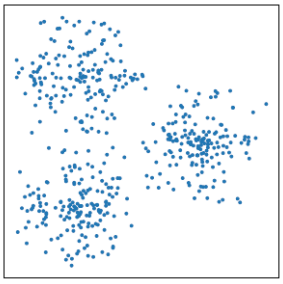
TP : ALGORITHME DES k MOYENNES

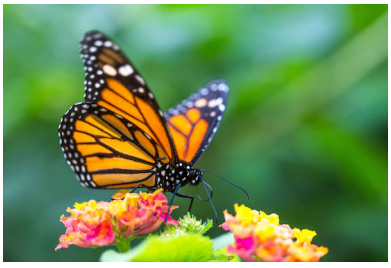
L'algorithme des k-moyennes, inventé par Lloyd en 1957, est un algorithme d'apprentissage non-supervisé. Il est fondé sur l'heuristique gloutonne. Il consiste à partir de données (non étiquetées), à classer ces données en k-classes, en regroupant les données proches au sein d'une même classe ; autrement dit, étant donné un ensemble S de données, l'algorithme calcule une partition à k sous-ensembles de S.

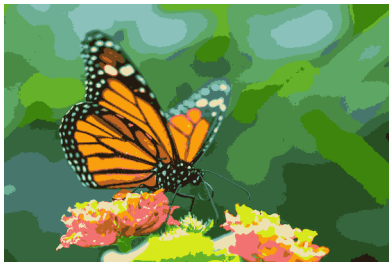
Par exemple, nous percevons tous trois classes parmi les points de la figure ci-contre. L'objectif de la machine est de faire ce regroupement de points (mais nous devons préciser à l'algorithme le nombre de classes à prendre en compte).



1 Un exemple : réduction des couleurs d'une image

Considérons l'image représenté ci-dessous. Ses dimensions sont de 415×626 pixels. Chaque pixel est un triplet (r, g, b) où r, g et b sont des flottants de l'intervalle $[0, 1[$ codés sur 32 bits représentant la quantité de rouge, vert et bleu du pixel. L'image est représentée en machine par un tableau numpy de taille $(415, 626, 3)$.





Pour pouvoir tester l' algorithme sur une image pas trop grande, vous pourrez utiliser le fichier : `papillon_reduit.png`.

Le fichier `TP_KMEANS.py` vous fournit une fonction qui permet de déterminer la liste des couleurs de l'image, sous la forme d'une liste de 3-uplets.
A partir de $415 \times 626 = 259790$ pixels, nous obtenons une liste de 121915 couleurs différentes.
Vous pouvez visualiser les 3 premières couleurs (sous forme de coordonnées (r,g,b)) de A en tapant `Scol(:3)`.

Notre objectif est de réduire le nombre de couleurs à seulement $k = 16$ couleurs. *(Il faudra dans un premier temps tester avec $k = 4$ pour éviter une trop longue attente).*
Pour ce faire, nous allons appliquer l'algorithme des k moyennes pour regrouper les différentes couleurs en k classes, calculer la couleur moyenne de chacune de ces classes, puis attribuer cette valeur moyenne à chacun des pixels de la classe correspondante.

2 Principe de l'algorithme des k-moyennes

Soit S un ensemble de points d'un espace vectoriel E muni d'une distance.

1. On choisit aléatoirement k points $c_0, \cdots, c_{k-1}$ parmi les points de S ; ces points constituent les *centres* initiaux.

2. chacun des points x_i de l'ensemble S est associé au centre c_j le plus proche de x_i ; on crée ainsi k classes $G_0, \cdots, G_{k-1}$: G_j est l'ensemble des points de S associés au centre c_j.
3. pour chaque classe G_j calculée à l'étape précédente, on détermine le barycentre de G_j par :

$$b_j = \frac{1}{\text{card}(G_j)} \sum_{x \in G_j} x$$

4. on remplace les centres $c_0, \cdots, c_{k-1}$ par les $b_0, \cdots, b_{k-1}$ et on reprend le calcul à partir de l'étape (2) tant que les nouveaux centres calculés b_j sont différents des c_j.

Nous admettrons que cet algorithme termine, autrement dit qu'à partir d'une certaine étape, les centres $c_0, \cdots, c_{k-1}$ ne se déplacent plus et donc que les classes $G_0, \cdots G_{k-1}$ sont stabilisées.

Cet algorithme regroupe les points proches car il renvoie un minimum (local) de

$$D(c_0, \cdots, c_k) = \sum_{j=0}^{k-1} \sum_{x \in G_j} \|x - c_j\|^2$$

Attention, rien nous assure d'obtenir une solution optimale (un minimum globale de D).

3 Implémentation de l'algorithme des k-moyennes

Nous allons écrire l'algorithme dans un cadre assez général.

3.1 Représentation des données

Dans tout ce TP, un point désigne un p-uplet (type `tuple`)
Nous représenterons

- l'ensemble des points donnés par une liste `S` de `tuple` :
  `S = [x0, x1, ... ] = [ (0.145, 0.541, 0.234), (0.230,0.650,0.700), ... ]`.
- l'ensemble des centres de chaque classe par une liste `Lcentres` de `tuple` :
  `Lcentres = [c0, c1, ... ] = [ (0.403,0.302,0.204) , (0.100,0.500,0.200) ...]`.
- une classe par un entier entre 0 et $k-1$;
- l'ensemble des k classes dans S, par une liste `Lclasses` de taille `n=len(S)`, d'entiers compris entre 0 et $k-1$, et tel que pour chaque i entre 0 et $n-1$, le point `S[i]` appartient à la classe `Lclasses[i]`, ce qui signifie que le centre le plus proche du point `S[i]` sera donné par `Lcentres[Lclasses[i]]`.
  Par exemple, si `S=[x0,x1,x2,x3]=[(1.1,2.5), (2.5,0.5), (1.4,0.9) ,(3.3,0.5)]`,
  `Lcentres=[c0,c1]=[(3,1) , (1.2,1.5)]` alors `Lclasses=[1,0,1,0]` car
  - le centre le plus proche de `x0` est `c1` -> `Lclasses[0]= 1`
  - le centre le plus proche de `x1` est `c0` -> `Lclasses[1] = 0`

3.2 Algorithme des k-moyennes

Nous importons pour ce TP, les modules suivants :

```
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as rd
```

1. Ecrire une fonction `dist(x,y)` qui calcule la distance euclidienne entre deux points x et y.

   Cette fonction ayant déjà été écrite au TP précédent, nous vous donnons une solution dans le fichier `TP_KMEANS.py`.

2. Ecrire une fonction `initialise(S,k)` qui à partir d'une liste de points S et d'un entier k, renvoie une liste de k points (distincts) choisis aléatoirement dans S.

   On pourra utiliser la fonction `rd.randint(vmin,vmax)` qui renvoie un entier compris entre `vmin` (inclus) et `max` (exclu).

3. Ecrire une fonction `plusProcheCentre(x,Lcentres)` qui à partir d'un point x et d'une liste de centres `Lcentres` renvoie l'élément de `Lcentres` le plus proche de x noté `cmin`, ainsi que l'indice `jmin` tel que `Lcentres[jmin] = cmin`.

4. Ecrire une fonction `calcul_classes(S,Lcentres)` qui à partir d'un ensemble de points `S` et d'une liste de centres `Lcentres`, renvoie la liste `Lclasses` de taille `len(S)`, telle que pour chaque point `S[i]` de `S`, `Lclasses[i]` est l'indice `jmin` tel que `Lcentres[jmin]` est le centre le plus proche de `S[i]` ; autrement dit, `Lclasses[i]` est le numéro du centre le plus proche de `S[i]`.

5. Ecrire une fonction `barycentre(P)` qui à partir d'un ensemble de points (sous la forme d'une liste de `tuple`) renvoie le barycentre de P : $b = \dfrac{1}{\text{len}(P)} \displaystyle\sum_{x \in P} x$. Cette fonction devra renvoyer un `tuple`.

   Attention, l'addition de `tuple` en Python correspond à la concaténation.

   Indication : il n'est pas pratique d'utiliser des `tuple` pour calculer un barycentre. Vous pourrez transformer chacun des points x de P en tableau numpy par la commande `np.array(x)` pour chaque p-uplets x dans P.

6. Ecrire une fonction `calcul_un_centre(S,Lclasses,j)` qui à partir de l'ensemble des points `S`, d'une liste `Lclasses` donnant la classe de chacun des points de `S` (voir question précédente) et d'un entier j, renvoie le barycentre des points de `S` appartenant à la classe j, c'est à dire les points `S[i]` de `S` tel que `Lclasses[i] = j`.

7. Ecrire une fonction `calcul_centres(S,Lclasses,k)` qui à partir des points de `S`, de la liste des classes `Lclasses` et un entier `k`, renvoie la liste des centres pour chaque classe j compris entre 0 et $k-1$.

8. L'algorithme des k-moyennes s'arrête lorsque deux listes de centres consécutives sont égales. Pour implémenter ce test d'arrêt, il nous faut pouvoir évaluer la distance entre deux listes de points L1 et L2. Nous utiliserons pour cela, le maximum des distances entre les points de L1 et L2 :

   `max([dist(L1[j], L2[j]) for j in range(k)])`.

   A l'aide des fonctions définies précédemment, écrire l'algorithme des k-moyennes `k_moyennes(S,k)` qui renvoie la liste des centres calculés `Lcentres`, ainsi que la liste des classes `Lclasses`.

4 Application à la réduction des couleurs

1. Pour $k = 4$, à partir de la liste des couleurs `S_col`, calculer la liste `Lcent_A` des k couleurs obtenues à partir de l'algorithme des k moyennes, ainsi que la liste `Lcla_A` donnant le numéro (entre 0 et $k-1$) de chaque couleur de `S_col`.

   Tracer, en utilisant que les composantes de rouge (en abscisse) et vert (en ordonnée) de chaque couleur, les 4 couleurs obtenues, ainsi qu'un échantillon de points (une centaine par exemple) de `S_col`.

   Vous ferez en sorte que votre échantillon de points contienne des points de chaque classe.

2. Ecrire une fonction `nouvelle_image(A,k)` qui à partir d'une matrice `A` et d'un entier `k`, renvoie une nouvelle matrice `B` où le pixel `B[i,j]` correspond à la couleur la plus proche calculée à partir de l'algorithme des k-moyennes.

   Pour afficher la matrice B, il suffira ensuite d'écrire :

```
B = nouvelle_image(A,k)
plt.imshow(B)
plt.show()
```