INF391 - Reconocimiento de Patrones en Minería de Datos

Tarea 1: Técnicas de *Clustering*

Francisca Ramírez

Juan Pablo Muñoz

17 de abril del 2019

Introducción

En esta tarea se exploran distintas técnicas de reconocimiento de patrones basadas en *clustering* vistas en cátedra. Para ello, se cuenta con tres pequeños *datasets* con distintas características, que servirán para contrastar la aptitud que cada técnica posee para cada caso.

Luego de la experimentación, se responden las dos preguntas conceptuales planteadas en el enunciado.

Parte I

Primero, se prepara la ingesta de datos.

In [1]:

```
import os.path
import numpy as np

def ingest_dataset(txt_dir):
    dataset = list()
    if os.path.exists(txt_dir):
        with open(txt_dir, 'r') as f:
            for line in f.readlines():
                data_point = line.split()
                x_coord, y_coord = float(data_point[0]), float(data_point[1])
                dataset.append([x_coord, y_coord])
    return np.array(dataset)
```

Y se instancian los tres datasets.

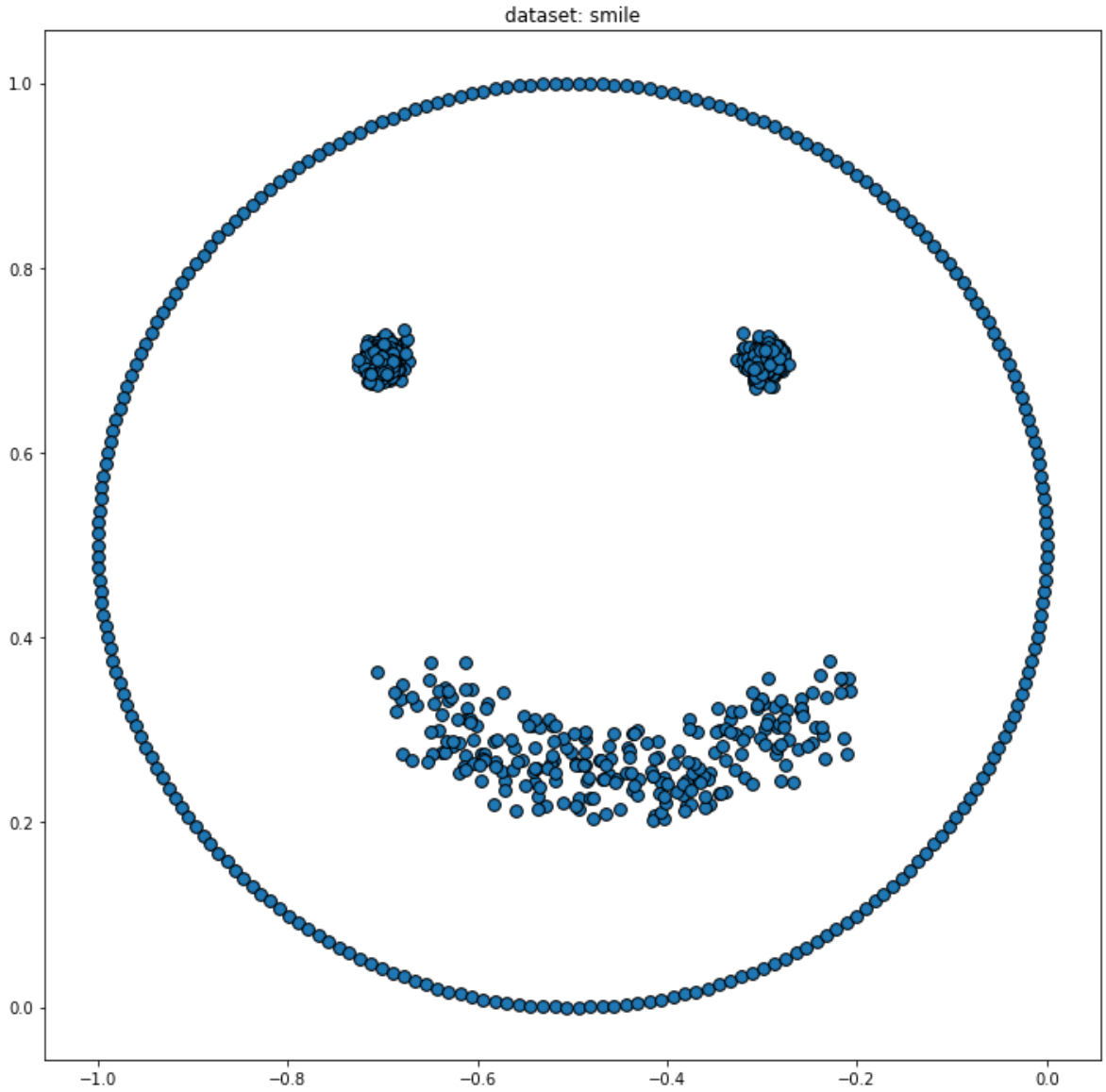
In [2]:

```
smile = ingest_dataset('smile.txt')
mouse = ingest_dataset('mouse.txt')
spiral = ingest_dataset('spiral.txt')
```

El hecho de que estos datasets sean 2-dimensionales nos permite visualizarlos fácilmente y obtener una clara idea cómo se distribuyen.

In [161]:

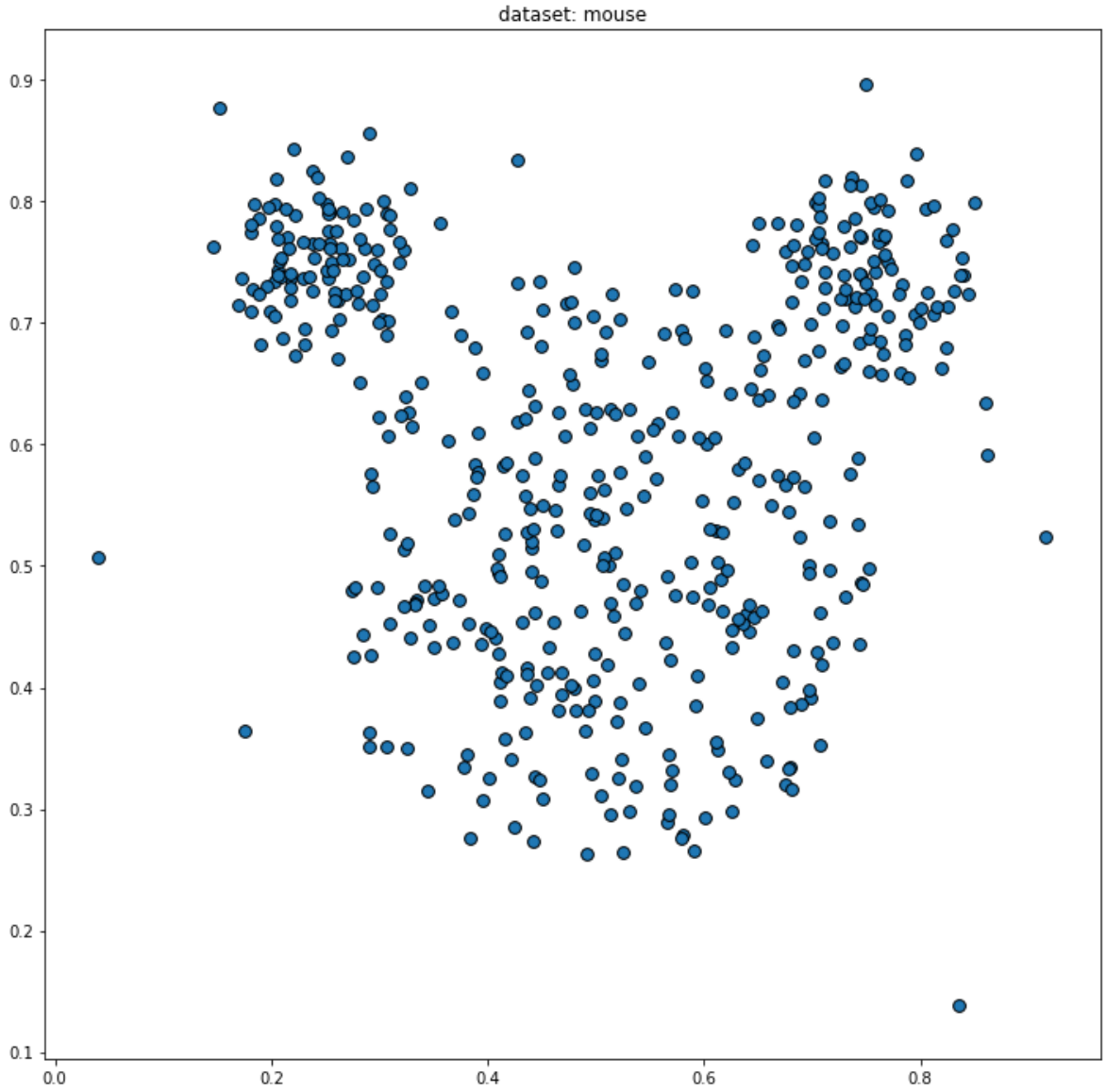
```
# Smile
plt.figure(figsize=(12,12))
plt.scatter(smile[:, 0], smile[:, 1], marker='o',
            edgecolors='k', s=60)
plt.title('dataset: smile')
plt.show()
```



Del gráfico, es evidente que este dataset posee 4 clusters: dos de ellos tienen una alta densidad y tienen forma convexa o globular (los ojos), uno tiene densidad media y es semi-convexo (la sonrisa) y el último es una distribución aparentemente uniforme en forma de circunferencia, que encierra el espacio que los tres clusters anteriores ocupan.

In [154]:

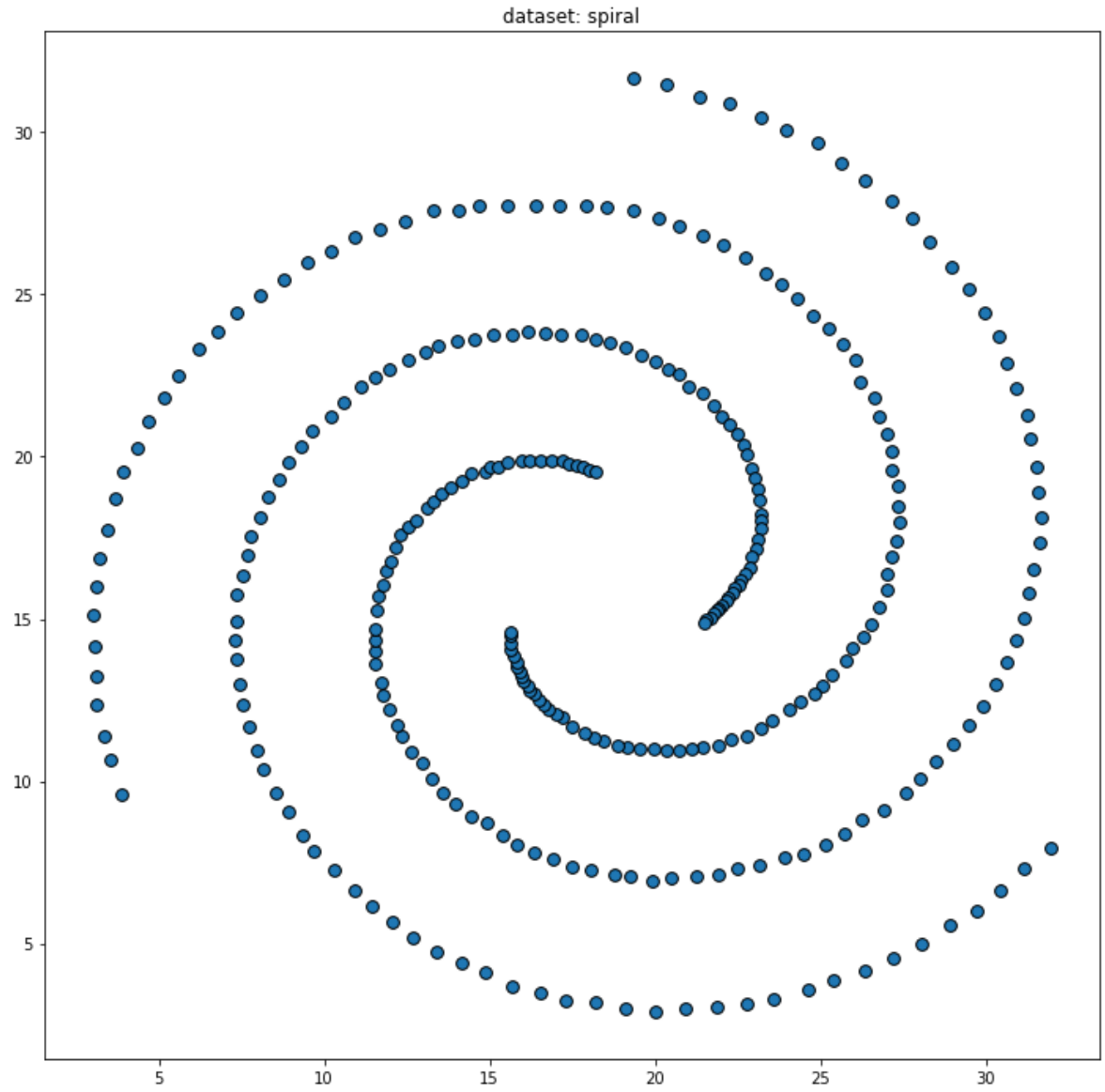
```
# Mouse
plt.figure(figsize=(12,12))
plt.scatter(mouse[:, 0], mouse[:, 1], marker='o',
            edgecolors='k', s=60)
plt.title('dataset: mouse')
plt.show()
```



Este dataset posee tres clusters circulares, uno de densidad mediana (cabeza) y dos más pequeños de densidad media-alta (orejas). Los tres clusters tienen forma convexa y tienen poca a nula intersección entre sí.

In [155]:

```
# Spiral
plt.figure(figsize=(12,12))
plt.scatter(spiral[:, 0], spiral[:, 1], marker='o',
            edgecolors='k', s=60)
plt.title('dataset: spiral')
plt.show()
```



La distribución de este dataset es no-convexa y resulta intuitivo distinguir cada espiral como un posible cluster distinto. En este caso, no es la densidad de los puntos lo que nos indica la presencia de un cluster, sino el patrón que estos describen.

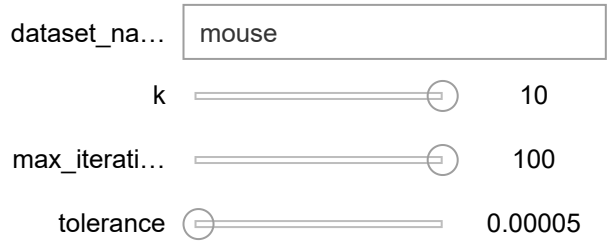
A continuación, se procede a aplicar las técnicas de *clustering*.

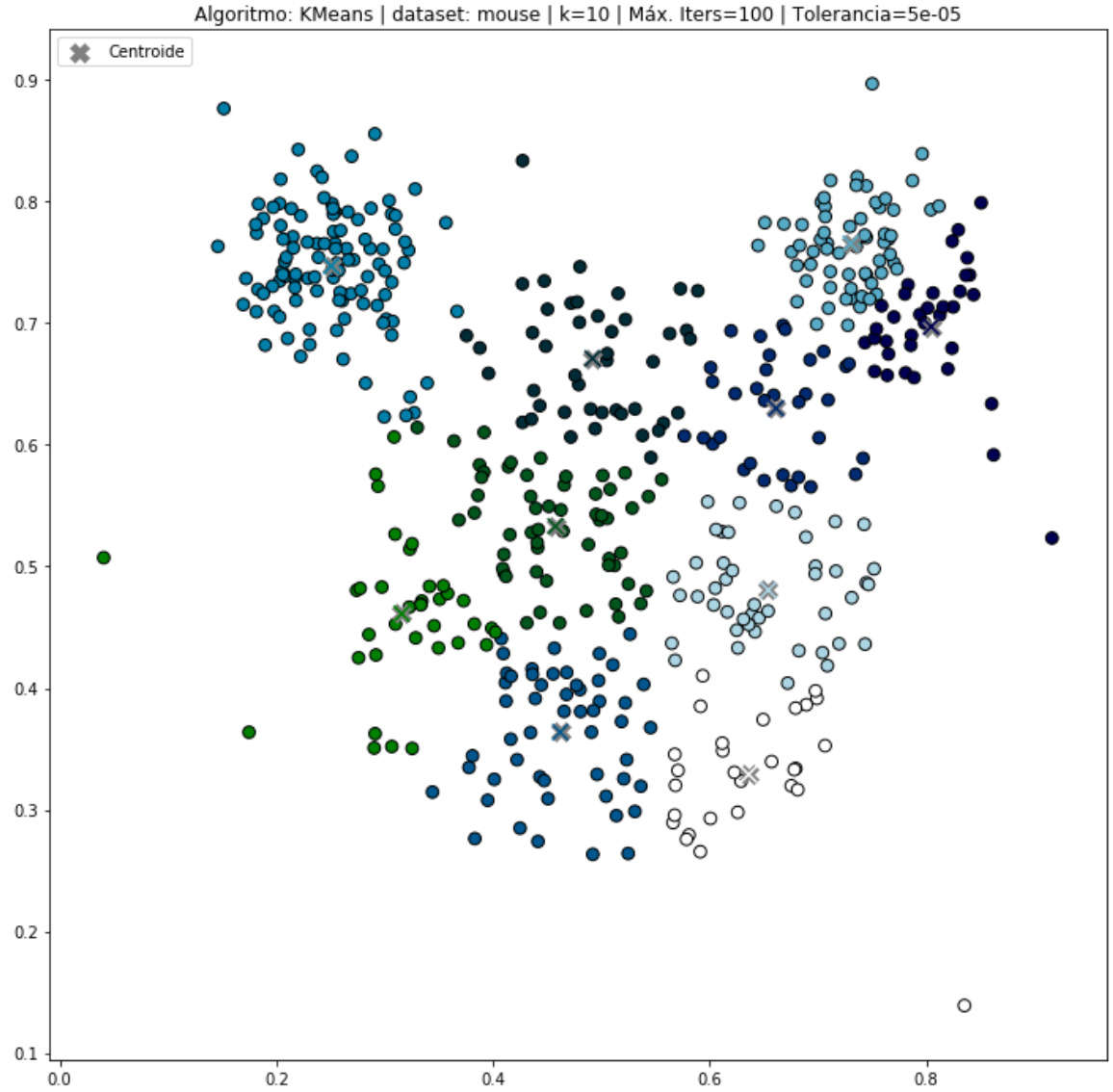
(Info teórica sobre los algoritmos de clustering excepto Fuzzy se puede hallar acá: [https://scikit-learn.org/stable/modules/clustering.html (https://scikit-learn.org/stable/modules/clustering.html)](https://scikit-learn.org/stable/modules/clustering.html))

1. K-Means

In [92]:

```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from ipywidgets import interact
from ipywidgets import FloatSlider

def apply_kmeans(dataset, k, max_iterations=300, tolerance=1e-4):
    kmeans = KMeans(
        n_clusters=k,
        init='random',
        n_init=1,
        max_iter=max_iterations,
        tol=tolerance,
        random_state=0,
    )
    kmeans.fit(dataset)
    return kmeans.cluster_centers_, kmeans.labels_

@interact(
    dataset_name=['smile', 'mouse', 'spiral'],
    k=(2,10, 1),
    max_iterations=(10, 100, 10),
    tolerance=FloatSlider(
        min=5e-5,
        max=5e-4,
        step=5e-5,
        continuous_update=False,
        readout=True,
        readout_format='.5f'
    ),
)
def plot_kmeans(dataset_name, k, max_iterations, tolerance):
    if dataset_name == 'smile':
        dataset = smile
    elif dataset_name == 'mouse':
        dataset = mouse
    elif dataset_name == 'spiral':
        dataset = spiral
    centroids, labels = apply_kmeans(dataset, k, max_iterations, tolerance)
    plt.figure(figsize=(12,12))
    plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c=labels,
                edgecolors='k', s=60, cmap=plt.cm.ocean)
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=150,
                linewidths=.5, c='gray', cmap=plt.cm.ocean, label='Centroide')
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=100,
                linewidths=2, c=list(range(len(centroids))),
                cmap=plt.cm.ocean)
    plt.title('Algoritmo: KMeans | dataset: {} | k={} | Máx. Iters={} | Tolerancia={}'
    plt.legend(loc='upper left')
```





Análisis K-Means

Bla...

---

2. Agglomerative Hierarchical Clustering

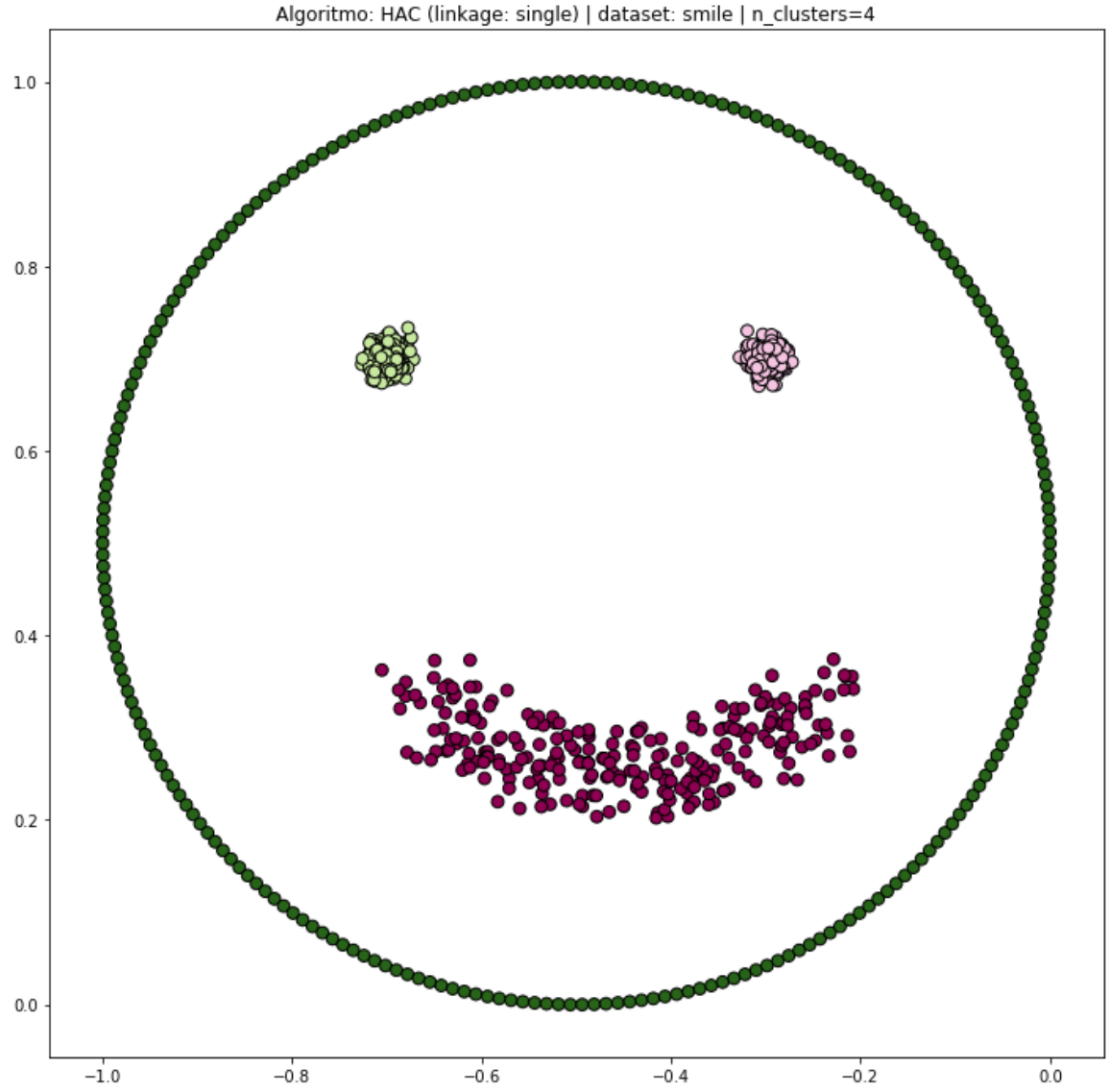
In [42]:

```
from sklearn.cluster import AgglomerativeClustering

def apply_hac(dataset, linkage, n_clusters):
    hac = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage)
    hac.fit(dataset)
    return hac.labels_

@interact(
    dataset_name=['smile', 'mouse', 'spiral'],
    linkage=['single', 'complete'],
    n_clusters=(2,10, 1),
)
def plot_hac(dataset_name, linkage, n_clusters):
    if dataset_name == 'smile':
        dataset = smile
    elif dataset_name == 'mouse':
        dataset = mouse
    elif dataset_name == 'spiral':
        dataset = spiral

    labels = apply_hac(dataset, linkage, n_clusters)
    plt.figure(figsize=(12,12))
    plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c=labels,
                edgecolors='k', s=60, cmap=plt.cm.PiYG)
    plt.title('Algoritmo: HAC (linkage: {}) | dataset: {} | n_clusters={}'.format(link
```

dataset_na… smile

linkage single

n_clusters 4



Análisis Agglomerative Hierarchical Clustering

Bla...

---

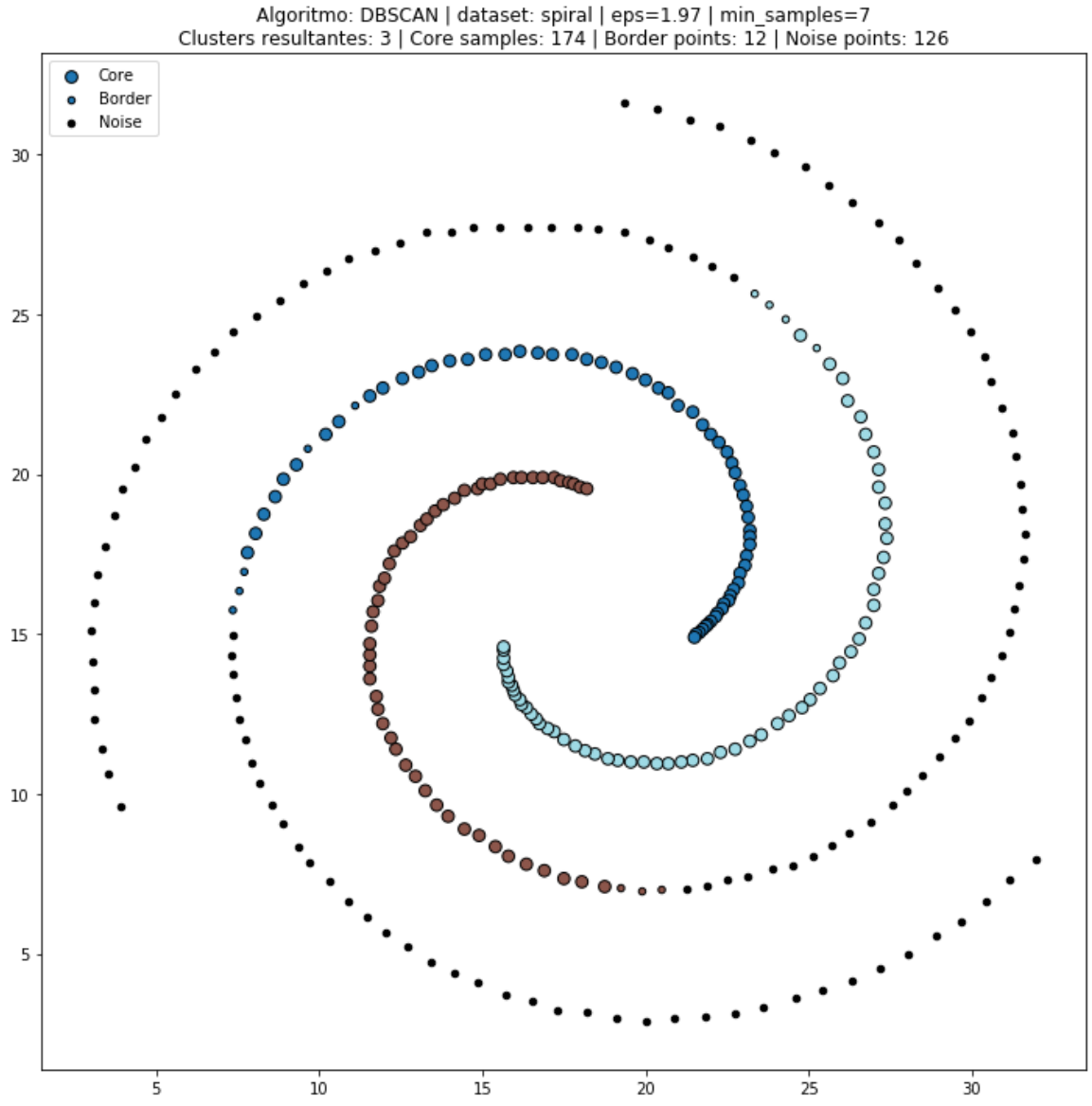
3. DBSCAN

In [62]:

```
from sklearn.cluster import DBSCAN

def apply_dbscan(dataset, min_pts, eps):
    dbscan = DBSCAN(eps=eps, min_samples=min_pts)
    dbscan.fit(dataset)
    core_samples_mask = np.zeros_like(dbscan.labels_, dtype=bool)
    core_samples_mask[dbscan.core_sample_indices_] = True
    noise_points_mask = (dbscan.labels_ == -1)
    border_points_mask = np.zeros_like(dbscan.labels_, dtype=bool)
    border_points_mask[~core_samples_mask & ~noise_points_mask] = True

    # Number of clusters in labels, ignoring noise if present.
    n_clusters_ = len(set(dbscan.labels_)) - (1 if -1 in dbscan.labels_ \
                                              else 0)
    n_noise_ = list(dbscan.labels_).count(-1)
    return dbscan.labels_, n_clusters_, n_noise_, core_samples_mask, \
        border_points_mask, noise_points_mask

@interact(
    dataset_name=['smile', 'mouse', 'spiral'],
    min_pts=(1,50, 1),
    eps=(0.01, 5.0, 0.01),
)
def plot_dbscan(dataset_name, min_pts, eps):
    if dataset_name == 'smile':
        dataset = smile
    elif dataset_name == 'mouse':
        dataset = mouse
    elif dataset_name == 'spiral':
        dataset = spiral
    labels, n_clusters, n_noise, core_samples_mask, border_points_mask,\
    noise_points_mask = apply_dbscan(dataset, min_pts, eps)
    core_points = dataset[core_samples_mask]
    border_points = dataset[border_points_mask]
    noise_points = dataset[noise_points_mask]
    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    plt.figure(figsize=(12,12))
    # Plot core samples
    plt.scatter(core_points[:, 0], core_points[:, 1], marker='o',
                c=labels[core_samples_mask], edgecolors='k', s=60,
                cmap=plt.cm.tab20, label='Core')
    # Plot border points
    plt.scatter(border_points[:, 0], border_points[:, 1], marker='o',
                c=labels[border_points_mask], edgecolors='k', s=20,
                cmap=plt.cm.tab20, label='Border')
    # Plot noise points
    plt.scatter(noise_points[:, 0], noise_points[:, 1], marker='o',
                c='black', edgecolors='k', s=20,
                cmap=plt.cm.tab20, label='Noise')
    plt.title('Algoritmo: DBSCAN | dataset: {} | eps={} | min_samples={}\n\
Clusters resultantes: {} | Core samples: {} | Border points: {} | Noise points: {}'
              .format(dataset_name, eps, min_pts, n_clusters,
                      len(core_points), len(border_points),
                      len(noise_points)))
    plt.legend(loc='upper left')

```

min_pts 7

eps 1.97



Análisis DBSCAN

Bla...

---

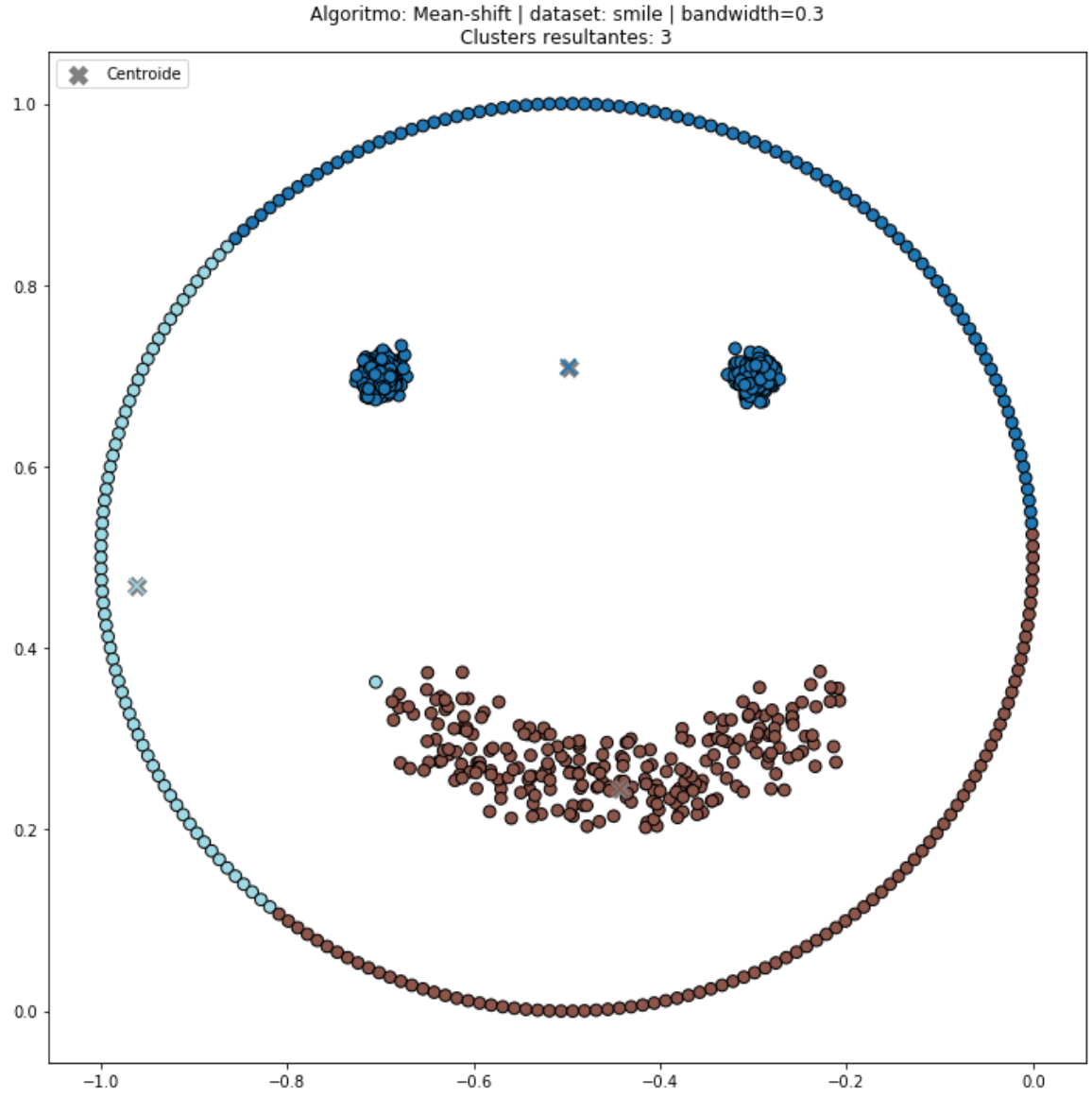
4. Mean-shift

In [67]:

```
from sklearn.cluster import MeanShift

def apply_meanshift(dataset, bandwidth):
    meanshift = MeanShift(bandwidth=bandwidth)
    meanshift.fit(dataset)
    return meanshift.cluster_centers_, meanshift.labels_

@interact(
    dataset_name=['smile', 'mouse', 'spiral'],
    bandwidth=(0.1, 10, 0.1),
)
def plot_kmeans(dataset_name, bandwidth):
    if dataset_name == 'smile':
        dataset = smile
    elif dataset_name == 'mouse':
        dataset = mouse
    elif dataset_name == 'spiral':
        dataset = spiral
    centroids, labels = apply_meanshift(dataset, bandwidth=bandwidth)
    plt.figure(figsize=(12,12))
    plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c=labels,
                edgecolors='k', s=60, cmap=plt.cm.tab20)
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=150,
                linewidths=.5, c='gray', cmap=plt.cm.tab20, label='Centroide')
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=100,
                linewidths=2, c=list(range(len(centroids))),
                cmap=plt.cm.tab20)
    plt.title('Algoritmo: Mean-shift | dataset: {} | bandwidth={}\nClusters resultantes
    plt.legend(loc='upper left')
```

dataset_na… smile

bandwidth 0.30



Análisis Mean-shift

Bla...

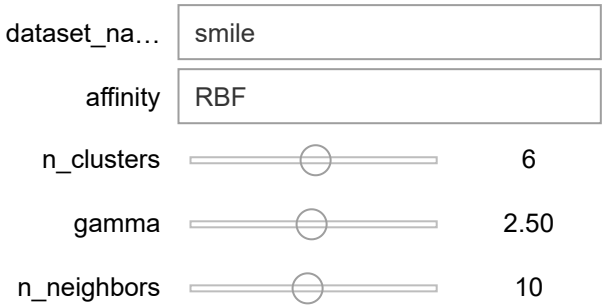
---

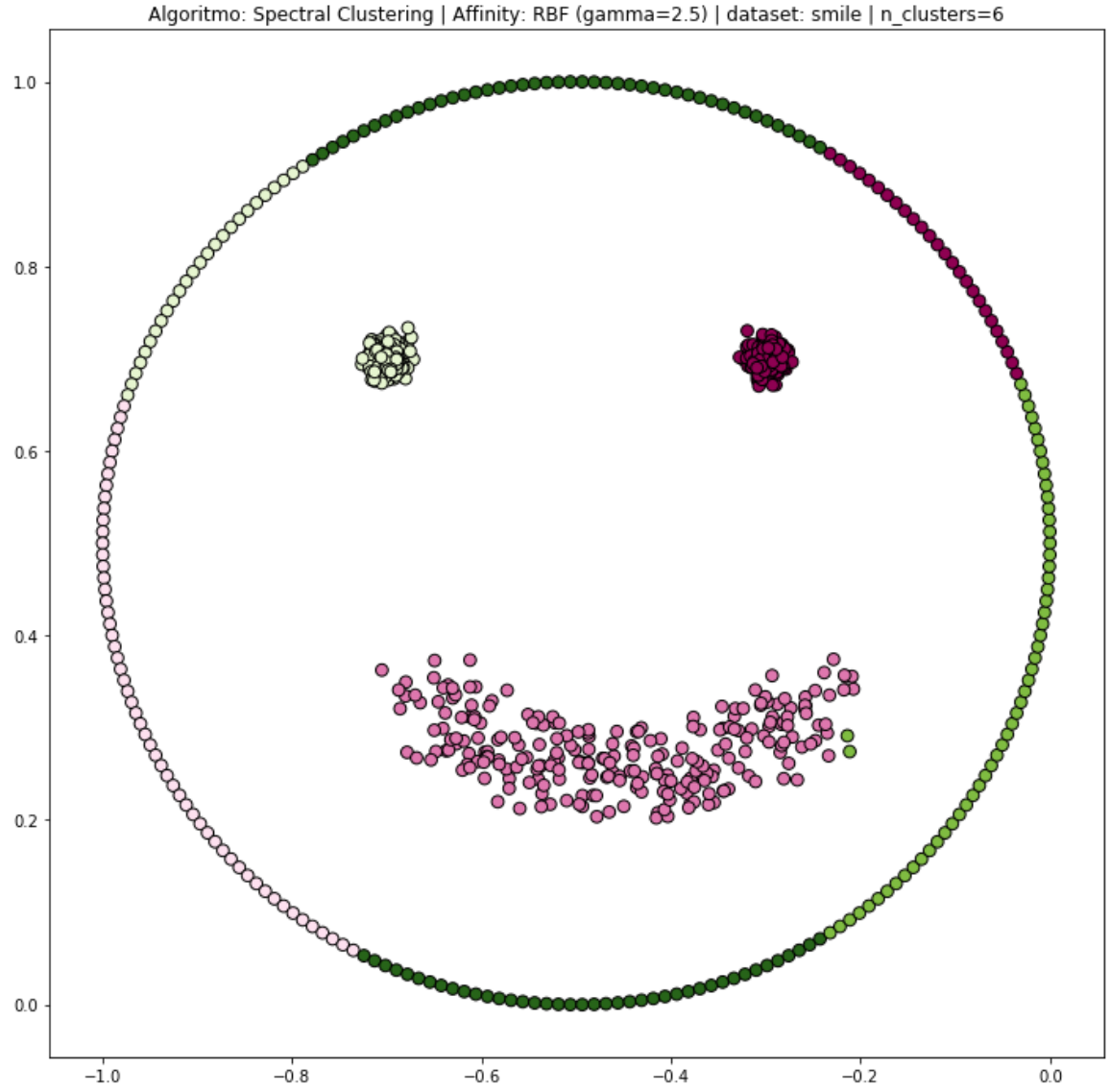
5. Spectral clustering

In [127]:

```
from sklearn.cluster import SpectralClustering

def apply_spectral_rbf(
    dataset,
    n_clusters,
    random_state=0,
    n_init=1,
    gamma=1.0,
    affinity_matrix_method='rbf',
):
    spectral = SpectralClustering(
        n_clusters=n_clusters,
        random_state=random_state,
        n_init=n_init,
        gamma=gamma,
        affinity=affinity_matrix_method,
    )
    spectral.fit(dataset)
    return spectral.labels_

def apply_spectral_nearest_neighbors(
    dataset,
    n_clusters,
    n_neighbors,
    random_state=0,
    n_init=1,
    affinity_matrix_method='nearest_neighbors',
):
    spectral = SpectralClustering(
        n_clusters=n_clusters,
        random_state=random_state,
        n_init=n_init,
        affinity=affinity_matrix_method,
        n_neighbors=n_neighbors,
    )
    spectral.fit(dataset)
    return spectral.labels_

@interact(
    dataset_name=['smile', 'mouse', 'spiral'],
    affinity=['RBF', 'K-nearest neighbors'],
    n_clusters=(2,10, 1),
    gamma=(0.1, 5.0, 0.1),
    n_neighbors=(1, 20, 1),
)
def plot_spectral(
    dataset_name,
    affinity,
    n_clusters,
    gamma,
    n_neighbors,
):
    if dataset_name == 'smile':
        dataset = smile
    elif dataset_name == 'mouse':
        dataset = mouse
    elif dataset_name == 'spiral':
        dataset = spiral
    if affinity == 'RBF':
```

```
        labels = apply_spectral_rbf(
            dataset=dataset,
            n_clusters=n_clusters,
            gamma=gamma,
            affinity_matrix_method='rbf',
        )
    elif affinity == 'K-nearest neighbors':
        labels = apply_spectral_nearest_neighbors(
            dataset=dataset,
            n_clusters=n_clusters,
            n_neighbors=n_neighbors,
            affinity_matrix_method='nearest_neighbors',
        )
    plt.figure(figsize=(12,12))
    plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c=labels,
                edgecolors='k', s=60, cmap=plt.cm.PiYG)
    if affinity == 'RBF':
        plt.title('Algoritmo: Spectral Clustering | Affinity: RBF\
(gamma={}) | dataset: {} | n_clusters={}'.format(round(gamma, 2),
                                                 dataset_name, n_clusters))
    elif affinity == 'K-nearest neighbors':
        plt.title('Algoritmo: Spectral Clustering | Affinity: {}-nearest\
neighbors | dataset: {} | n_clusters={}'.format(n_neighbors, dataset_name,
                                                 n_clusters))

```





(Se pueden agregar más métodos de construcción de matriz de afinidad)

Análisis Spectral clustering

Bla...

---

6. Fuzzy C-Means

In [145]:

```
from skfuzzy import cluster as fuzzy
from ipywidgets import IntSlider

# Info: https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_cmeans.html
def apply_cmeans(dataset, c, m, error, maxiter, seed=0):
    # El argumento 'data' de cmeans exige que el dataset venga transpuesto!
    cntr, u, u0, d, jm, p, fpc = fuzzy.cmeans(
        data=dataset.T,
        c=c,
        m=m,
        error=error,
        maxiter=maxiter,
        seed=seed,
    )
    return cntr, u, u0, d, jm, p, fpc

@interact(
    dataset_name=['smile', 'mouse', 'spiral'],
    n_clusters=(2, 10, 1),
    p_exponent=(1.1, 3.0, 0.1),
    error=FloatSlider(
        min=5e-4,
        max=5e-2,
        step=5e-4,
        continuous_update=False,
        readout=True,
        readout_format='.4f'
    ),
    max_iterations=(1, 1000, 1),
)
def plot_cmeans(
    dataset_name,
    n_clusters,
    p_exponent,
    error,
    max_iterations,
):
    if dataset_name == 'smile':
        dataset = smile
    elif dataset_name == 'mouse':
        dataset = mouse
    elif dataset_name == 'spiral':
        dataset = spiral
    cntr, u, u0, d, jm, p, fpc = apply_cmeans(
        dataset=dataset,
        c=n_clusters,
        m=p_exponent,
        error=error,
        maxiter=max_iterations,
    )
    # El color de cada punto es asignado segun el cluster al cual pertenezca
    # con mayor porcentaje
    labels = u.argmax(axis=0)
    plt.figure(figsize=(12,12))
    plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c=labels,
                edgecolors='k', s=60, cmap=plt.cm.ocean)
    plt.scatter(cntr[:, 0], cntr[:, 1], marker='X', s=150,
                linewidths=.5, c='gray', cmap=plt.cm.ocean, label='Centroide')
    plt.scatter(cntr[:, 0], cntr[:, 1], marker='x', s=100,
```

```
                linewidths=2, c=list(range(n_clusters)),
                cmap=plt.cm.ocean)
    plt.title('Algoritmo: Fuzzy C-Means | dataset: {} | n_clusters={} | \
p={} | Máx. Iters={} | error={}\nFuzzy partition coefficient (FPC): {}'.format(
        dataset_name,
        n_clusters,
        round(p_exponent, 2),
        max_iterations,
        round(error, 5),
        round(fpc, 3),
    ))
    plt.legend(loc='upper left')
```

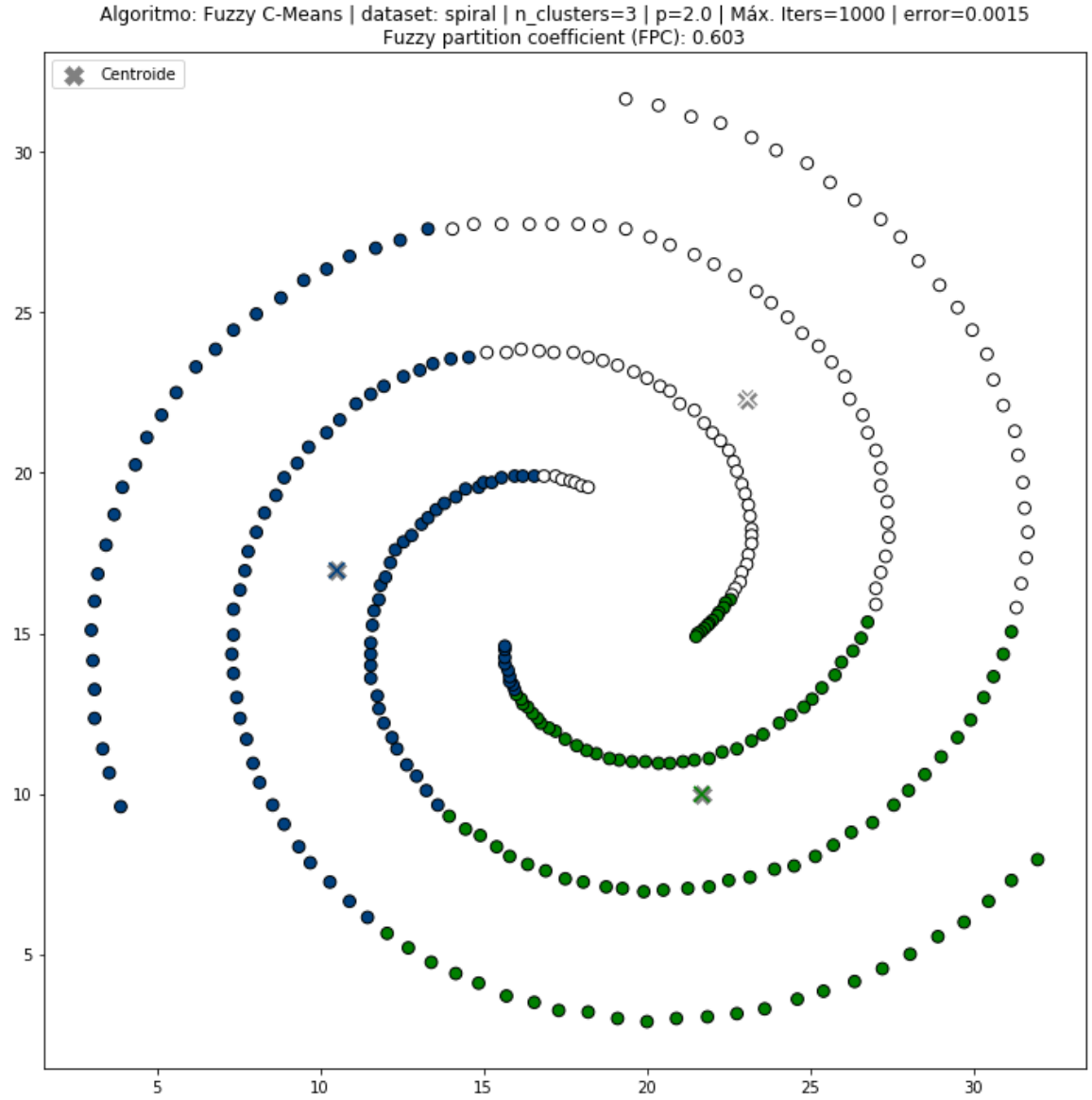
dataset_na… spiral

n_clusters 3

p_exponent 2.00

error 0.0015

max_iterati… 1000



Análisis Fuzzy C-Means

(este link puede ayudar con el análisis: https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_cmeans.html (https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_cmeans.html))

Bla...

Parte II

(a) Se tiene un conjunto de datos con 100 objetos. Se le pide realizar clustering utilizando K-means, pero para todos los valores de k, $1 \leq k \leq 100$, el algoritmo retorna que todos los clusters estan vacíos, excepto uno. ¿En que situación podría ocurrir esto? (analice los datos y no los parametros del algoritmo, i.e., iteraciones). ¿Qué resultado tendría single-link y DBSCAN para este tipo de datos?

```
1 Resp.:
```

(b) Considerando single-link y complete-link hierarchical clustering, ¿es posible que un objeto esté más cerca (en distancia Euclidiana) de los objetos de otros clusters en relación a los de su propio cluster? Si fuese posible, ¿en que enfoque (single y/o complete) esto podría ocurrir? Justifique con un ejemplo en cada caso.

Resp.: En cada iteración de single-link, un punto se incluye en cualquiera sea el cluster al cual pertenezca su punto más cercano. Al construirse los clusters de esta manera, no existe forma de que algún par de puntos de distintos clusters sean más cercanos entre sí que con los puntos de sus respectivos clusters.

En cambio, en complete-link esto sí puede ocurrir. Considerar, por ejemplo, la siguiente distribución de los objetos A, B, C, D, E en una dimensión:

```
A-B-C--D---E
```

Donde la distancia es directamente proporcional a la cantidad de carácteres entre los objetos. Si queremos un resultado de 2 clusters, se obtiene:

```
[A-B-C]--[D---E]
```

De aquí notamos que D dista de C (que está en un cluster distinto) sólo en 2 unidades, mientras que la distancia con E (que está en el mismo cluster) es de 3. Así, con este ejemplo, queda demostrada la posibilidad de que un objeto de un cluster puede tener como objetos más cercanos a aquellos de otros clusters.

In [ ]:

```
1
```