Homework 8 - Spectral embedding and clustering

This homework will not only help you get a better sense of how spectral embedding and clustering work, but also give you a stronger intuition for the "radial basis function" kernel, which is a common kernel used in many scientific applications.

Make sure you have downloaded:

- X1.csv
- X2.csv
- X3.csv

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

0. Intro

Remember:

- when performing spectral clustering, there is an adjacency matrix, which represents the graph/network of all data points.
- Data points are connected if they are "close enough" to each together.
- **Question:** But what constitutes close enough?
- There are two ways to do this, as seen in the lecture notes on Embedding:
  - k-nearest neighbors
  - $\exp\left(-d^2/\epsilon\right)$ similarity: radial basis function (RBF) proximity.

You know how k-nearest neighbors works. So this homework will start with introducing basic understanding of RBF and how it measures proximity. After that, we will make some functions that streamlines your code and experiment with spectral clustering models on synthetically generated data.

1. Radial Basis Function (RBF)

The RBF kernel, a.k.a. squared exponential/Gaussian kernel, is a function that takes in two points and outputs a number to reflect the proximity of those two points.

More formally, if the two points are x_1 and x_2, the "proximity" of these points are

$$k(x_1, x_2) \ := \ \exp\left(-\frac{\|x_1 - x_2\|^2}{2\gamma^2}\right),$$

where γ is a lengthscale parameter you can choose.

(γ *essentially* corresponds to the ϵ in the lecture notes and should remind you of the variance of a normal distribution PDF.)

The next task is to code up plot for RBF kernel, both in 1d and 2d.

But before we do that, we make a simplifying trick to make these plots easier.

- Notice that we are only concerned with the **distance** between two points, as represented in the $\|x_1 - x_2\|$ term in the formula above.
- Hence, we can set distance $d = \|x_1 - x_2\|$ to get a new formula for RBF kernel:

$$k(d) \ := \ \exp\left(-\frac{d^2}{2\gamma^2}\right),$$

1.1 RBF Function

Task:

[1 pt] complete the code for the function `rbf()`

- `d` is an array of distances
  - if d is 1d, you may think of it as the x-axis. So, d[i] is the distance from 0: $\sqrt{(x-0)^2} = |x-0|$
  - if d is 2d, you may think of it as the xy-plane. So, d[i,j] is the distance from the origin (0,0): $\sqrt{((x-0)^2 + (y-0)^2}$
- `gamma` is the lengthscale parameter

In [ ]:
```
def rbf(d,gamma):
    return np.exp(-(d**2) / (2 * gamma**2))
    # return None
```

1.2 1d plot

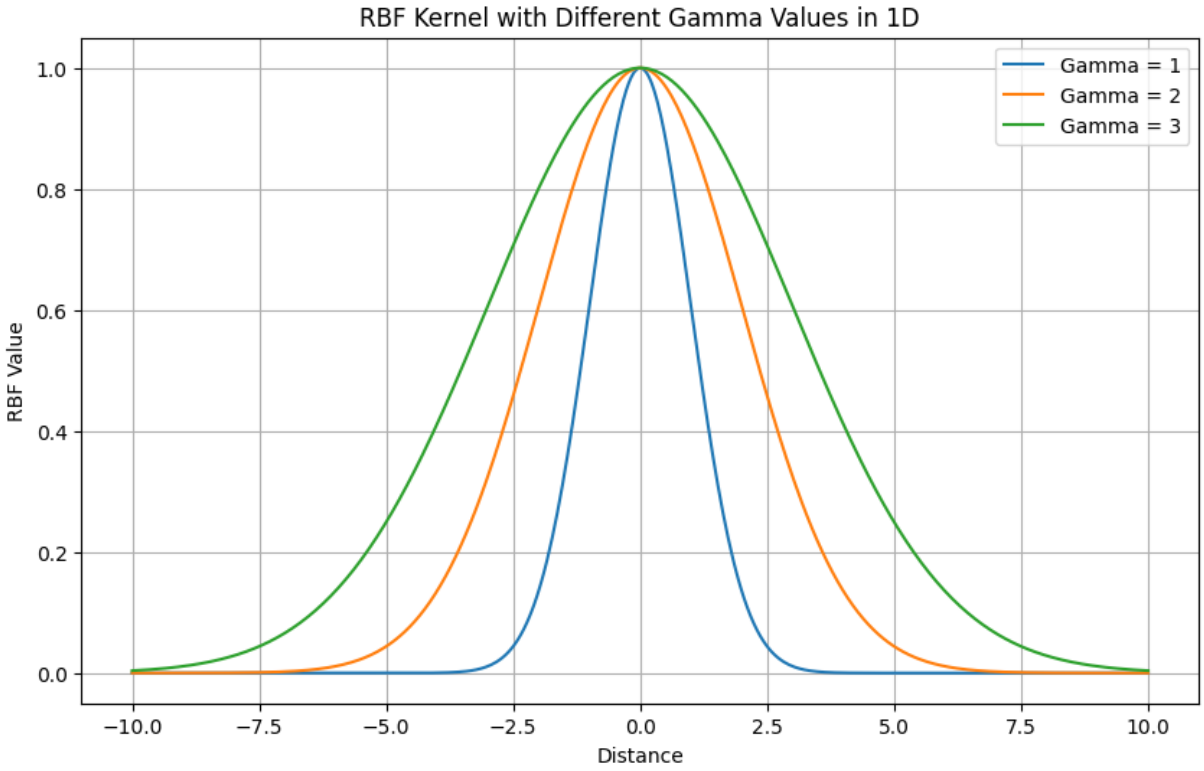
Task:

[2 pt] On the same figure and axes, plot rbf kernel for 1d case with multiple gamma.

- set x-axis to be from [-10,10]
- y-axis represent $k(x, 0)$
- set gamma = 1,2,3
- include title, axis labels, gridlines and a legend

In [ ]:
```
# TODO plot

d = np.linspace(-10, 10, 400)
```

```
gammas = [1, 2, 3]

plt.figure(figsize=(10, 6))
for gamma in gammas:
    plt.plot(d, rbf(d, gamma), label=f'Gamma = {gamma}')
plt.title('RBF Kernel with Different Gamma Values in 1D')
plt.xlabel('Distance')
plt.ylabel('RBF Value')
plt.legend()
plt.grid(True)
plt.show()
```



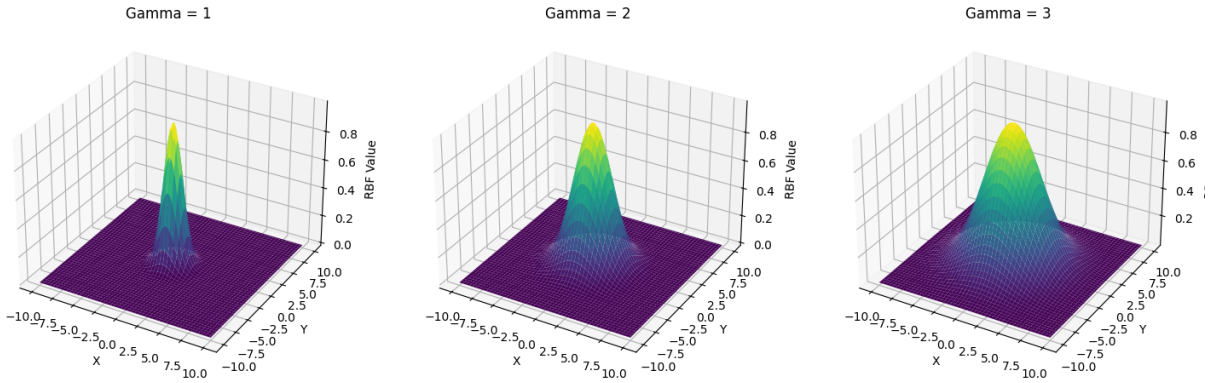
1.3 2d plot

Task:

[2 pt] On *3 separate* axes, plot rbf kernel for 2d case with multiple gamma.

- your plot will be a surface
  - you may need to use np.meshgrid()
  - you may use fig.add_subplot(1,3,i,projection='3d') to make three plots in a row. You can refer to this matplotlib tutorial for how to use it.
- set x-axis and y-axis to be both from [-10,10]
- z-axis will represent $k((x, y), (0, 0))$
- set gamma = 1,2,3
- include titles on each plot to make clear which gamma is being plotted and axis labels

In [ ]:
```
# TODO plot
x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)
D = np.sqrt(X**2 + Y**2)

fig = plt.figure(figsize=(18, 6))
for i, gamma in enumerate(gammas, 1):
    ax = fig.add_subplot(1, 3, i, projection='3d')
    Z = rbf(D, gamma)
    ax.plot_surface(X, Y, Z, cmap='viridis')
    ax.set_title(f'Gamma = {gamma}')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('RBF Value')
plt.show()
```



2. Spectral embedding/clustering functions

Before completing the spectral clustering experiments, complete the functions below.

Task:

[3 pt] complete code below for `embed_and_plot()`

- arguments, which will all be fed into the sklearn.manifold.SpectralEmbedding object:
  - `X`, data, number of samples by number of dimensions
  - `aff`, 'rbf' or 'nearest_neighbors' (short for affinity argument in SpectralEmbedding)
  - `gam`, gamma parameter for RBF kernel
  - `num_neighbors`, parameter for k-nearest neighbors
  - You can set default values for gam and num_neigbors to make your function calls later easier. Here is an example of how to write default parameters.
- your code should have two components
  1. make SpectralEmbedding object, fit it to data X, and obtain eigenvectors of embedding.
  2. three plots in the same row. (They should look)

    - scatter plot of embedding coordinates (eigenvector 1 against eigenvector 2)
    - scatter plot of embedding coordinates (eigenvector 2 against eigenvector 3)
    - scatter plot of sorted entries of eigenvector 1 (sorted index i against entry i)
- as usual, your plots should contain appropriate titles, axis labels, etc.
- You should read the SpectralEmbedding documentation **carefully**, so as to not make mistakes when implementing. Useful skill for any coding activity.

In [ ]:
```
from sklearn.manifold import SpectralEmbedding

"""
def embed_and_plot(X, aff, gam=1, num_neighbors=1):
    # TODO make model, fit, and obtain eigenvectors


    plt.figure(figsize=(18,4))
    plt.subplot(131)
    # TODO scatter eigenvector 1 entries against eigenvector 2 entries

    plt.subplot(132)
    # TODO scatter eigenvector 2 entries against eigenvector 3 entries

    plt.subplot(133)
    # TODO scatter sorted eigenvector 1 entries

    plt.show()
"""

def embed_and_plot(X, aff, gam=1, num_neighbors=5):
    embedder = SpectralEmbedding(n_components=3, affinity=aff, gamma=gam, n_
    X_embedded = embedder.fit_transform(X)

    plt.figure(figsize=(18, 4))

    # Eigenvector 1 vs Eigenvector 2
    plt.subplot(131)
    plt.scatter(X_embedded[:, 0], X_embedded[:, 1])
    plt.title('Eigenvector 1 vs 2')
    plt.xlabel('Eigenvector 1')
    plt.ylabel('Eigenvector 2')

    # Eigenvector 2 vs Eigenvector 3
    plt.subplot(132)
    plt.scatter(X_embedded[:, 1], X_embedded[:, 2])
    plt.title('Eigenvector 2 vs 3')
    plt.xlabel('Eigenvector 2')
    plt.ylabel('Eigenvector 3')

    # Sorted Eigenvector 1
    plt.subplot(133)
    plt.scatter(range(len(X)), np.sort(X_embedded[:, 0]))
    plt.title('Sorted Eigenvector 1')
    plt.xlabel('Index')
    plt.ylabel('Eigenvector 1 value')
```

```
    plt.show()
```

Task:

[2 pt] complete code below for `cluster_and_plot`

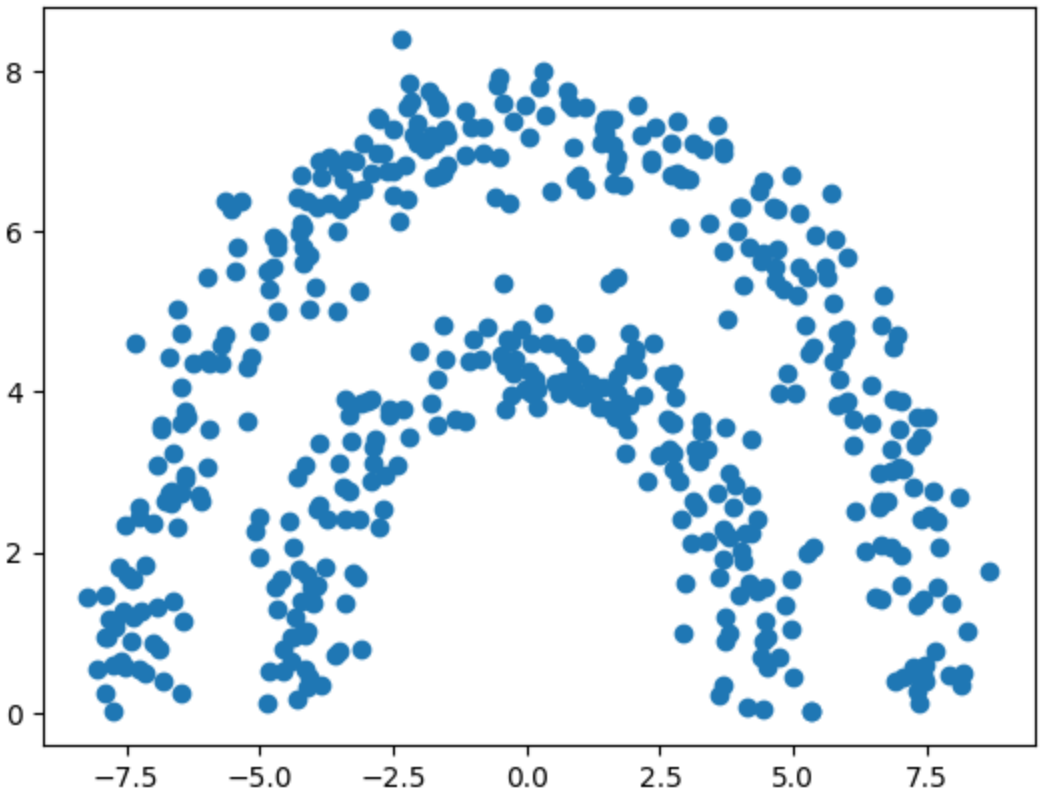
- arguments:
  - `X`, `aff`, `gam`, `num_neighbors` as in embed_and_plot
  - `n_clus` is the number of clusters that you want to split the data X into.
- your code should have two components:
  1. make SpectralClustering object, fit to data X, and predict the cluster labels.
  2. scatter plot of data with the predicted cluster labels. Include a title.

In [ ]:
```
from sklearn.cluster import SpectralClustering

"""
def cluster_and_plot(X, n_clus, aff, gam=1, num_neighbors=1):
    # TODO make model, fit, get cluster labels

    # TODO plot

    plt.show()
"""

def cluster_and_plot(X, n_clus, aff, gam=1, num_neighbors=5):
    clusterer = SpectralClustering(n_clusters=n_clus, affinity=aff, gamma=ga
    labels = clusterer.fit_predict(X)

    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
    plt.title(f'Spectral Clustering Results: {n_clus} Clusters')
    plt.show()
```

3. Cluster X1

Load and scatter plot `X1`.

In [ ]:
```
X1 = np.loadtxt('X1.csv', delimiter=',')
plt.scatter(X1[:,0], X1[:,1]); plt.show()
```

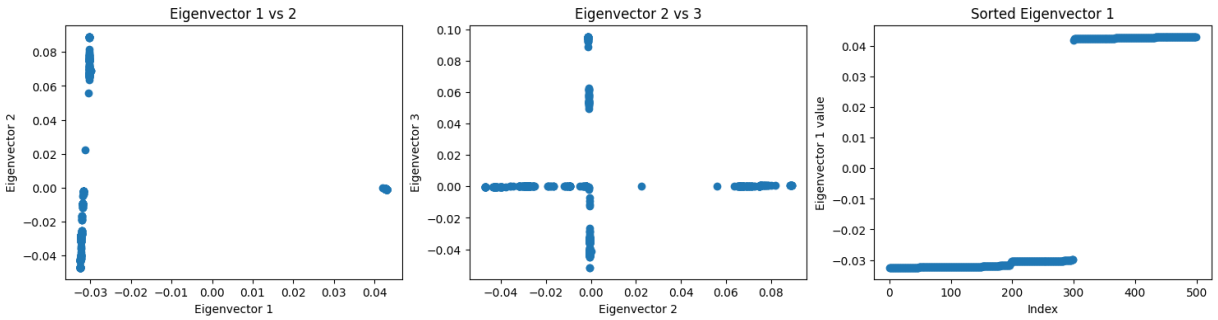


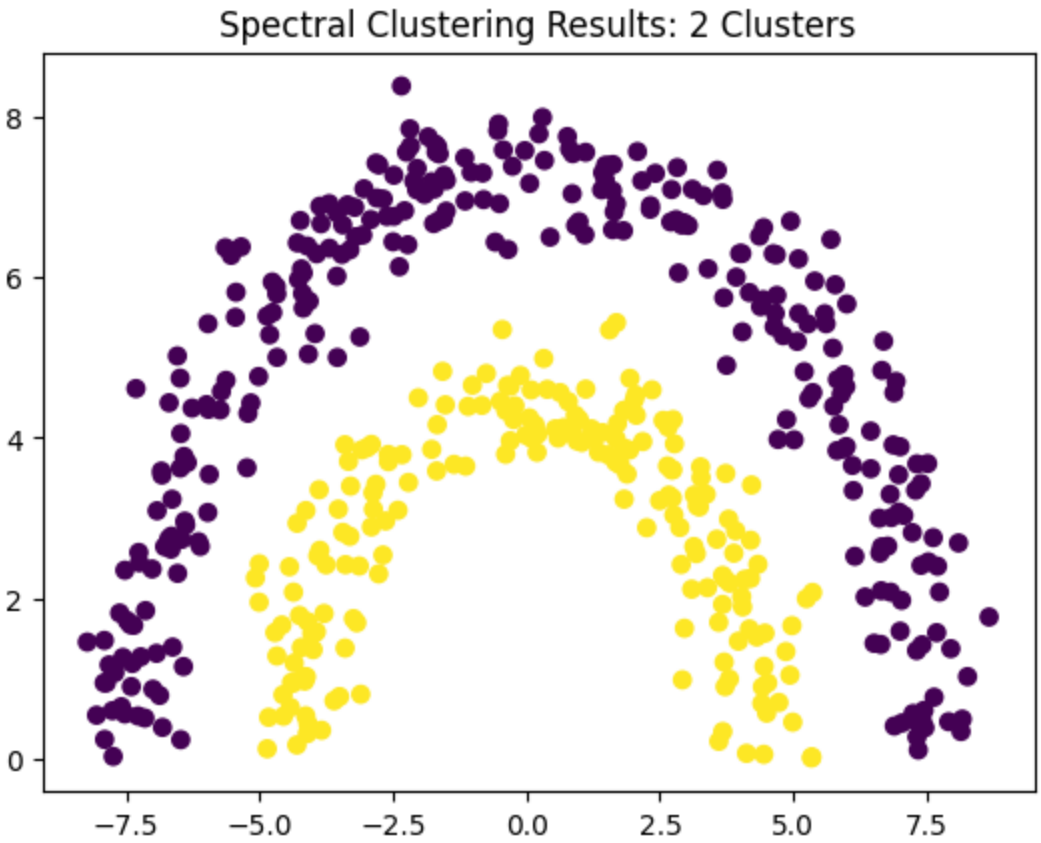
Task:

[1 pt] call embed_and_plot and cluster_and_plot on X1.

- you should correctly cluster the data into the two obvious clusters.
- experiment with different choices of affinity (rbf/nearest_neighbors) and parameter values (gamma/n_neighbors)
- all plots should be visible.

In [ ]:
```
# TODO

# Embed and Plot
embed_and_plot(X1, aff='rbf', gam=15, num_neighbors=20)

# Cluster and Plot
cluster_and_plot(X1, n_clus=2, aff='rbf', gam=15, num_neighbors=20)
```

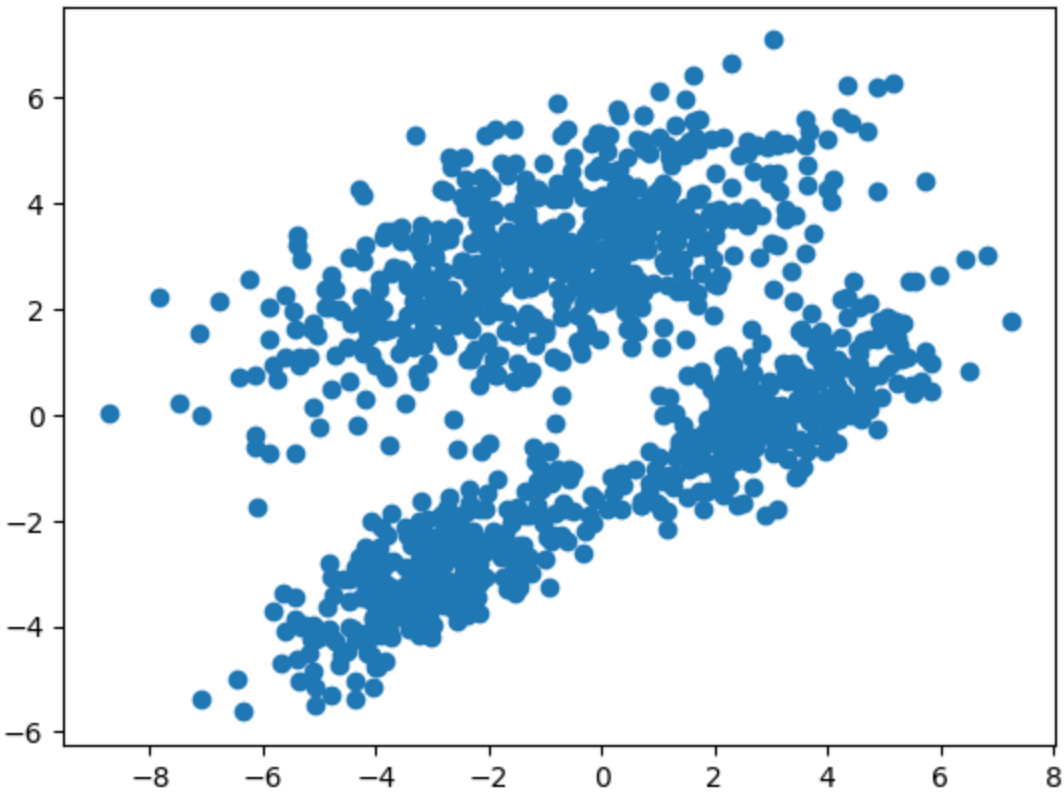




4. Cluster X2

Load and scatter plot `X2`.

In [ ]:
```
X2 = np.loadtxt('X2.csv', delimiter=',')
plt.scatter(X2[:,0], X2[:,1]); plt.show()
```

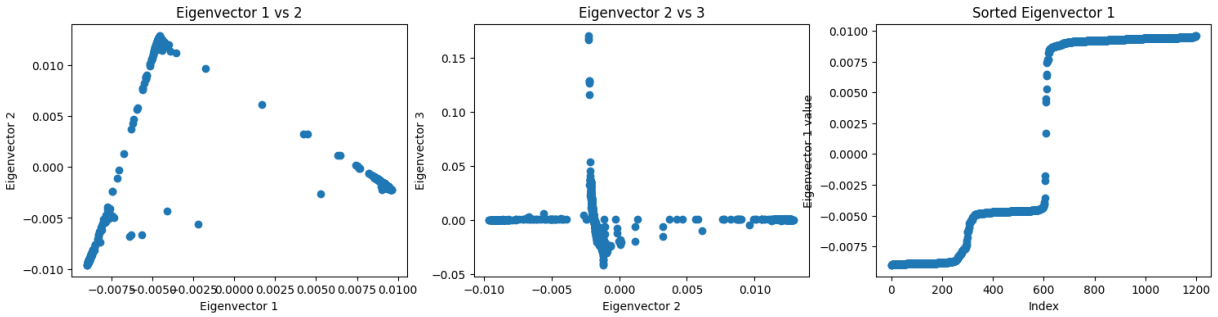


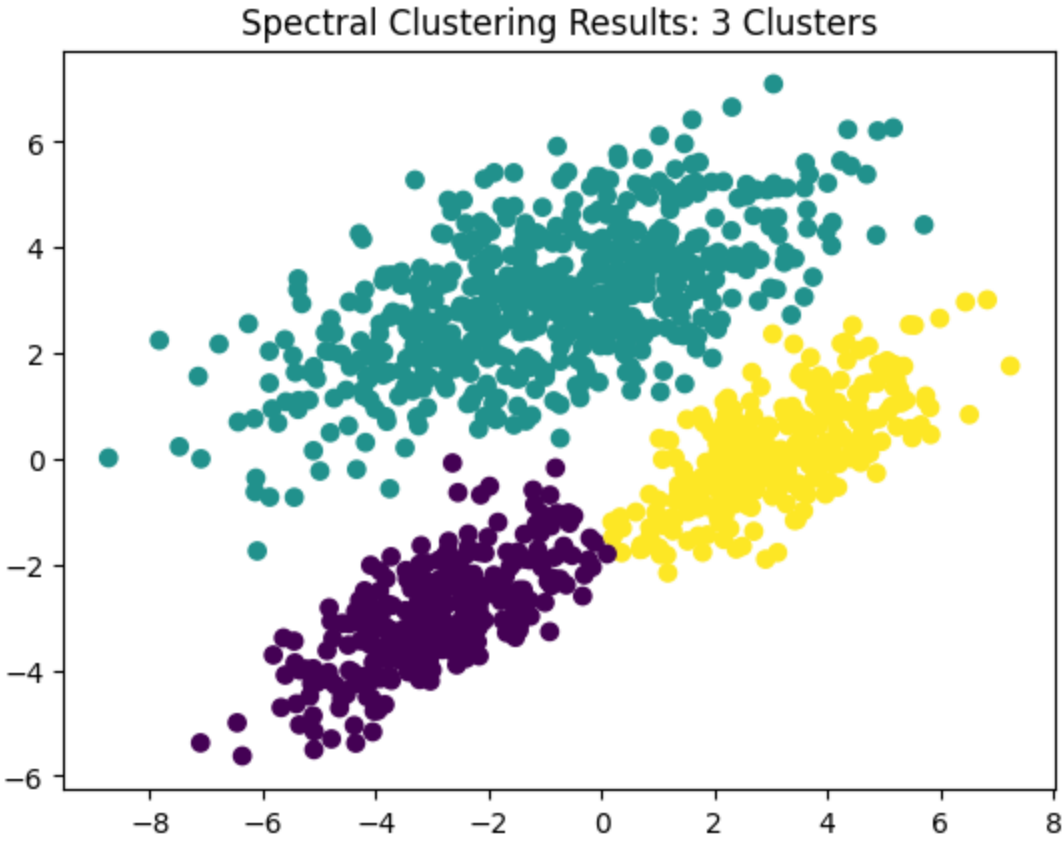
4.1 cluster

[1 pt] call embed_and_plot and cluster_and_plot on X2.

- same instructions as X1, except that you should obtain 3 clusters.

In [ ]:
```
# TODO

# Embed and Plot
embed_and_plot(X2, aff='rbf', gam=5, num_neighbors=5)

# Cluster and Plot for 3 clusters
cluster_and_plot(X2, n_clus=3, aff='rbf', gam=5, num_neighbors=5)
```





4.2 Compare to k-means and GMM

Task:

1. [2 pt] Think back to other clustering algorithms in this course. Between k-means and GMM, which is more approapriate for X2 data?

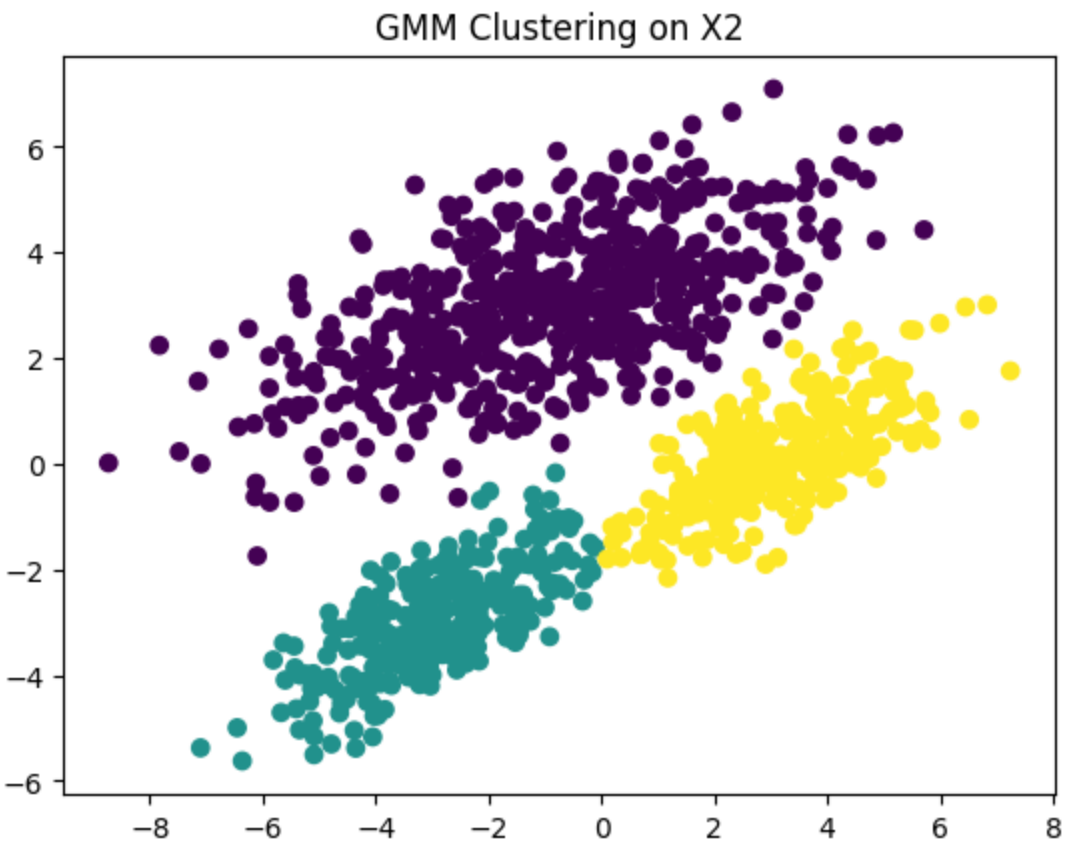
   - Give a reason why your chosen clustering algorithm is suited the data.
   - Give a reason why clustering algorithm you did not choose has a limitation that prevents it from being suitable to the data.

   Ans: GMM is more appropriate than k-means for this graph. GMM is more flexible in cluster covariance, so clusters can be elongated and possibly non-spherical as in this data set since the clusters are pills. K-means assumes spheres with similar sizes because of its centroid differences, which will lead to issues with this data set since you'd have some parts in the top cluster (turquoise) turn purple and yellow if they distance-wise close to that cluster.

2. [2 pt] Implement the clustering algorithm that you chose in the previous question to be appropriate for X2 data.

   - You may use relevant sklearn packages.
   - Include a plot of the cluster labels to ensure that it is working exactly to how you expect.

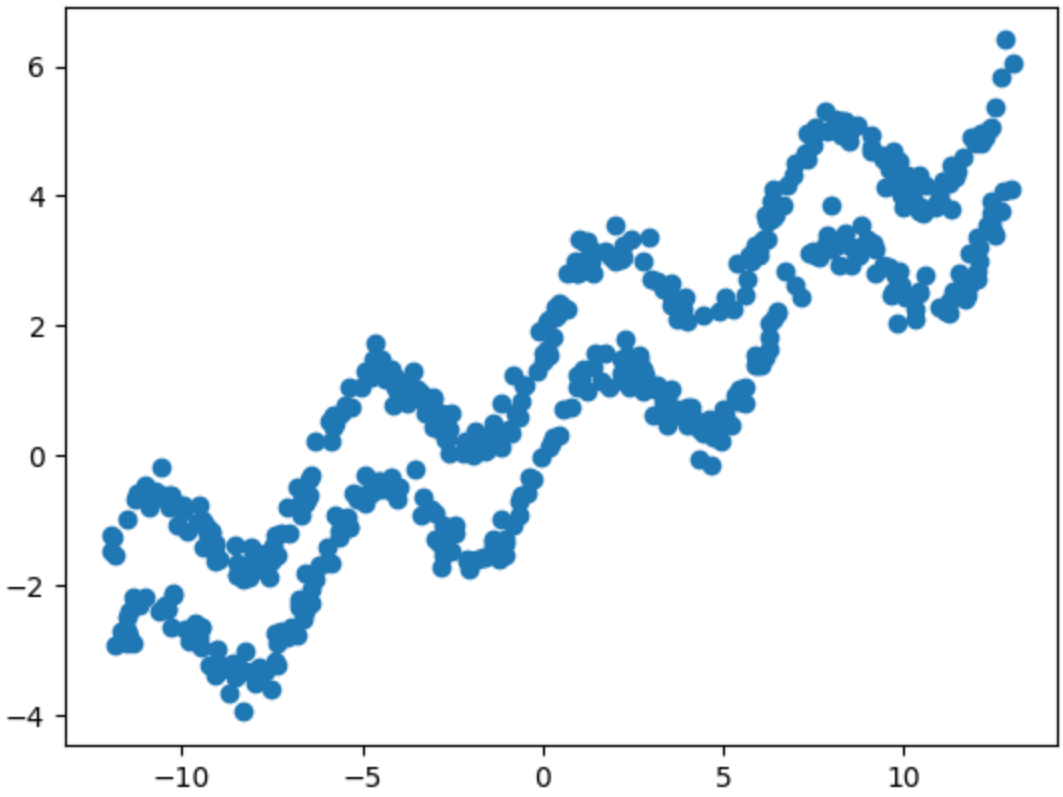
In [ ]:
```
# TODO implement clustering based on your answer

from sklearn.mixture import GaussianMixture

# Load X2 data
X2 = np.loadtxt('X2.csv', delimiter=',')

# Implement GMM
gmm = GaussianMixture(n_components=3, covariance_type='full')
gmm_labels = gmm.fit_predict(X2)

# Plot
plt.scatter(X2[:, 0], X2[:, 1], c=gmm_labels, cmap='viridis')
plt.title('GMM Clustering on X2')
plt.show()
```



5. Cluster X3

Load and scatter plot `X3`.

In [ ]:
```
X3 = np.loadtxt('X3.csv', delimiter=',')
plt.scatter(X3[:,0], X3[:,1]); plt.show()
```

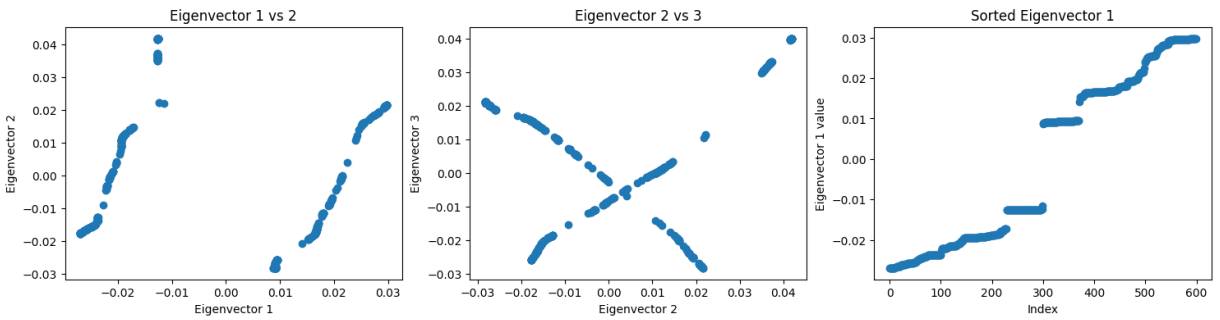


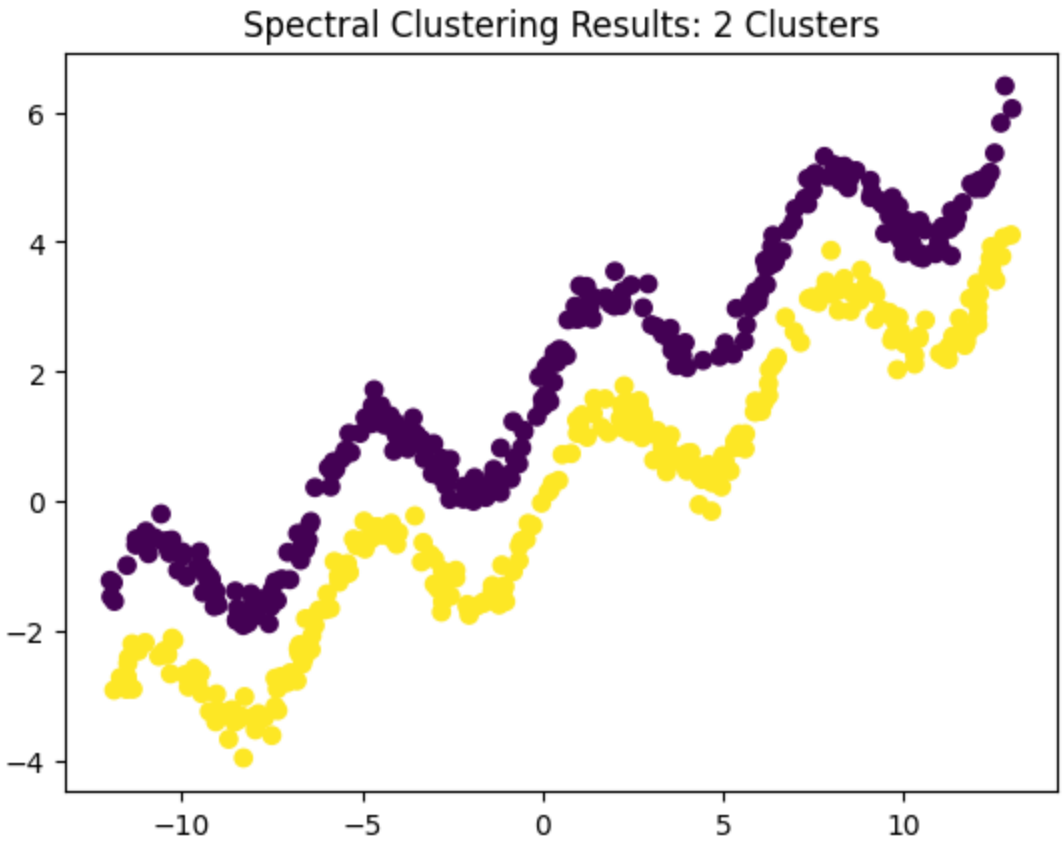
5.1 Spectral embedding/clustering

[1 pt] call embed_and_plot and cluster_and_plot on X3.

- Same instructions as for X1 and X2.

In [ ]:
```
# TODO

# Embed and Plot
embed_and_plot(X3, aff='rbf', gam=10, num_neighbors=30)

# Cluster and Plot
cluster_and_plot(X3, n_clus=2, aff='rbf', gam=10, num_neighbors=30)
```



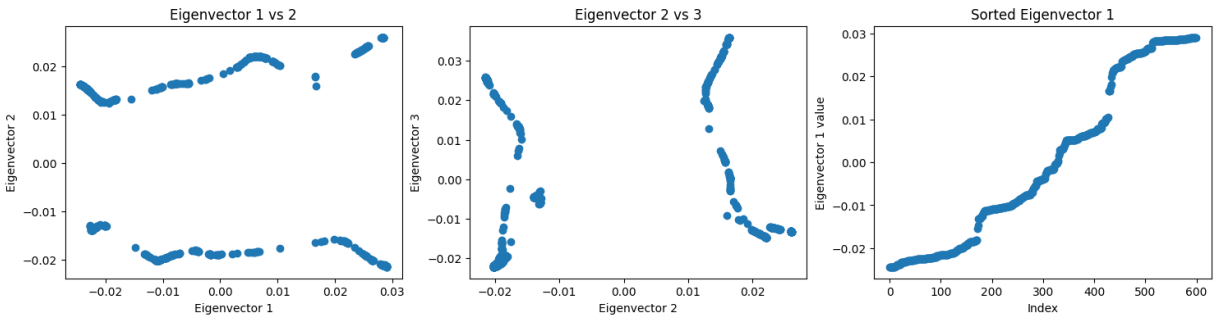


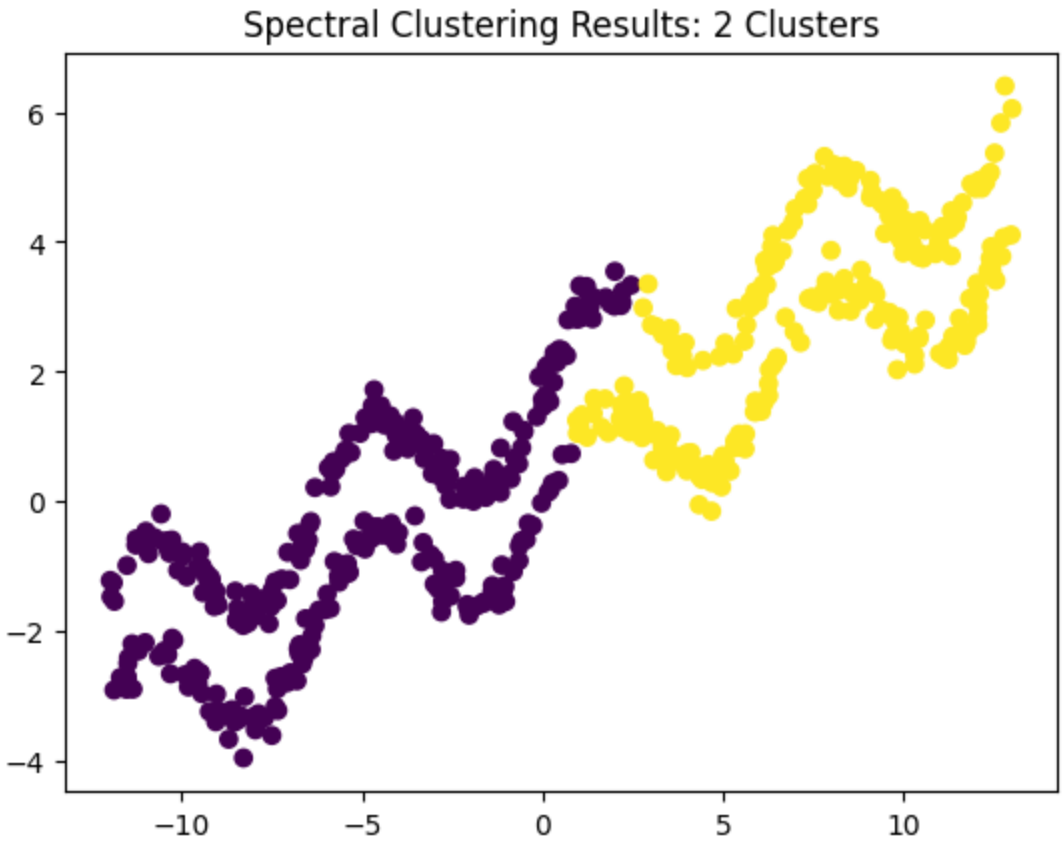
5.2 When sklearn "doesn't work"

Fix aff='rbf' and gam=8.

Call embed_and_plot and cluster_and_plot on X3. (You should get a failed clustering)

In [ ]:
```
# TODO Try aff='rbf' and gam=8

# Embed and Plot
embed_and_plot(X3, aff='rbf', gam=8, num_neighbors=5)

# Cluster and Plot
cluster_and_plot(X3, n_clus=2, aff='rbf', gam=8, num_neighbors=5)
```





In [ ]:

In [ ]:

Your intervention to make spectral clustering still work

The sklearn package with spectral clustering doesn't seem to work with parameters as fixed above (aff='rbf' and 'gam=8'). So, we need to go beyond default package implementations and make some adjustments to show that the overarching idea of spectral clustering still works.

Task:

1. [2 pt] What do you observe about the points scattered in the *embedding* plots for X1, X2, and X3 when the spectral clustering method successfully clusters the data? Explain how the spectral embedding plots suggests to us that the spectral embedding method should still be applicable in clustering X3 data with these fixed parameter values.

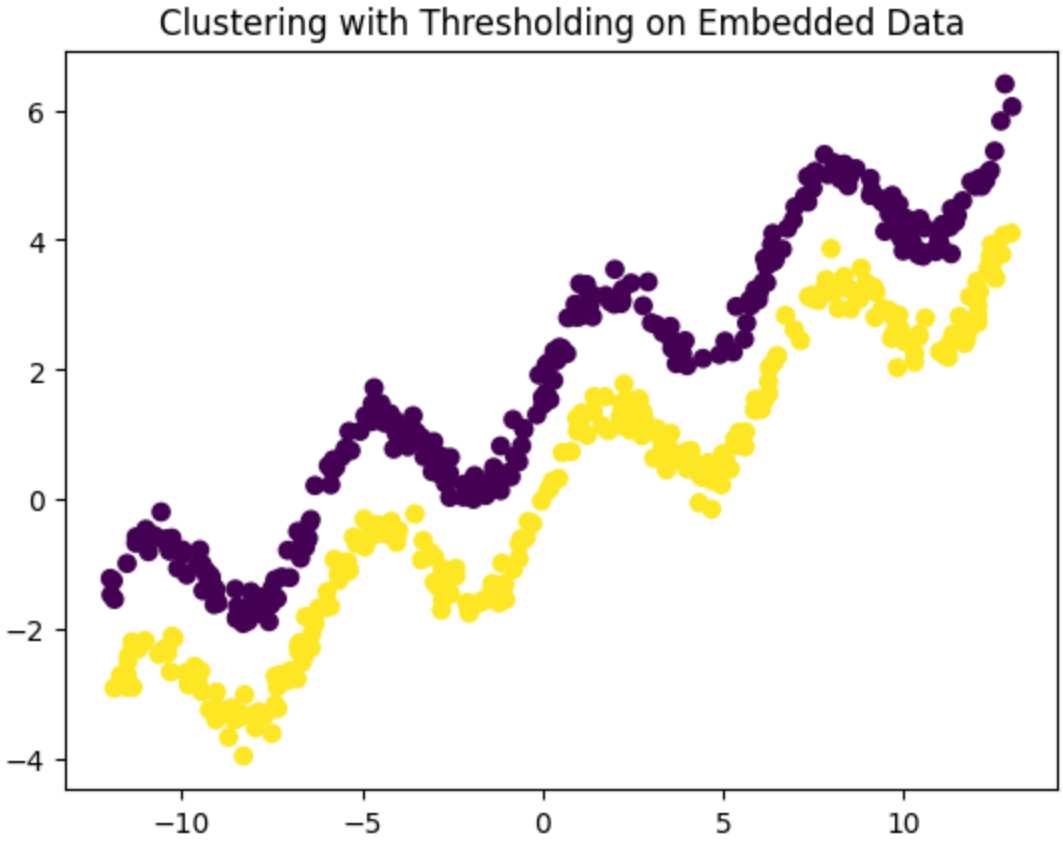
   Ans: Other successful clustering for X1, X2, and X3 had good clustering of points along eigenvector 1. With these parameters, the eigenvector 1 vs 2 doesn't show well-separated structures along the eigenvector 1 axis. However, eigenvector 2 vs 3 does, hinting that at a higher dimension (eigenvector 2) in the embedding there could be

more meaningful patterns we could use for clustering. Thus, we should try using the second eigenvector with a simple thresholding.

2. [1 pt] Make an adjustment to the clustering process in order to successfully cluster the data.

    - You do not have to call the functions we wrote earlier. You can write it up yourself, using any relevant sklearn packages.
    - Make sure to plot the successful clustering results.
    - Hint: You can use the same spectral embedding, but perform a different clustering process on the embedded data.
    - Hint: Use the second eigenvector, not the first! A simple thresholding suffices for the intent of this homework.
        - However, to stretch your skills/get practice, you may perform a clustering algorithm (k-means/GMM) on the embedded data instead of the simple thresholding suggested above. It's just a few lines of code.

In [ ]:
```
# TODO make spectral clustering work :)

embedder = SpectralEmbedding(n_components=3, affinity='rbf', gamma=8, n_neig
X3_embedded = embedder.fit_transform(X3)

# Simple thresholding on the second eigenvector
threshold = np.median(X3_embedded[:, 1])  # Using median as a simple thresho
labels_threshold = (X3_embedded[:, 1] > threshold).astype(int)

# Plot with simple thresholding
plt.scatter(X3[:, 0], X3[:, 1], c=labels_threshold, cmap='viridis')
plt.title('Clustering with Thresholding on Embedded Data')
plt.show()
```


Clustering with Thresholding on Embedded Data
6
4
2
0
−2
−4
−10
−5
0
5
10