Visualization_Ex_08.ipynb

June 16, 2024

Shyam Kumar Ray Yadav (18449154)

Sonu Kumar (15651099)

Ahmad Raza Khawaja (25848862)

1 Problem sheet 8

1.1 Exercise 8.1: intrinsic dimension and spectral embedding with the Laplace operator.

In this problem we study in more detail the apparent intrinsic dimension of point data with the spectral embedding via the Laplace operator.

1. First we generate an example dataset. Let $n_1 = 25$, $n_2 = 20$. Let X be a set of $n_1 \cdot n_2$ points in $\mathbb{R}^2$, lying on a regular two-dimensional Cartesian grid with n_1 and n_2 points along the two axes, with distance 1 between points along each axis. So X should be an array of shape $(n_1 \cdot n_2) \times 2$. Generate X and compute the matrix D of squared Euclidean distances between the points in X. D should be of shape $(n_1 \cdot n_2) \times (n_1 \cdot n_2)$.

[213]:
```
import numpy as np
from scipy.linalg import eigh
import matplotlib.pyplot as plt
from PIL import Image
```

[172]:
```
#Generate an example dataset.

n1 = 25
n2 = 20
x = np.linspace(0, n1-1, n1) #points on X-axis
y = np.linspace(0, n2-1, n2) #Points on Y-axis

#Generate a mesh grid of n1 points for the X-axis and n2 points for the Y-axis.
xx, yy = np.meshgrid(x, y)
print(xx, "\n", yy)

xx_ravel = xx.ravel() #Changing n1 grid points of X-axis from 2D to 1D.
yy_ravel = yy.ravel() #Changing n2 grid points of Y-axis from 2D to 1D.
```

```python
X = np.vstack([xx_ravel, yy_ravel]).T #Stack 1D array as rows in 2D array and␣
↪transform.
print(f'The shape of pints lying on 2D Cartesian Grid:{X.shape}')
```

```
[[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
  18. 19. 20. 21. 22. 23. 24.]]
 [[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
   0.  0.  0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
   1.  1.  1.  1.  1.  1.  1.]
```

```
 [ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.
   2.  2.  2.  2.  2.  2.  2.]
 [ 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
   3.  3.  3.  3.  3.  3.  3.]
 [ 4.  4.  4.  4.  4.  4.  4.  4.  4.  4.  4.  4.  4.  4.  4.  4.  4.  4.
   4.  4.  4.  4.  4.  4.  4.]
 [ 5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.
   5.  5.  5.  5.  5.  5.  5.]
 [ 6.  6.  6.  6.  6.  6.  6.  6.  6.  6.  6.  6.  6.  6.  6.  6.  6.  6.
   6.  6.  6.  6.  6.  6.  6.]
 [ 7.  7.  7.  7.  7.  7.  7.  7.  7.  7.  7.  7.  7.  7.  7.  7.  7.  7.
   7.  7.  7.  7.  7.  7.  7.]
 [ 8.  8.  8.  8.  8.  8.  8.  8.  8.  8.  8.  8.  8.  8.  8.  8.  8.  8.
   8.  8.  8.  8.  8.  8.  8.]
 [ 9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.  9.
   9.  9.  9.  9.  9.  9.  9.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
  10. 10. 10. 10. 10. 10. 10.]
 [11. 11. 11. 11. 11. 11. 11. 11. 11. 11. 11. 11. 11. 11. 11. 11. 11. 11.
  11. 11. 11. 11. 11. 11. 11.]
 [12. 12. 12. 12. 12. 12. 12. 12. 12. 12. 12. 12. 12. 12. 12. 12. 12. 12.
  12. 12. 12. 12. 12. 12. 12.]
 [13. 13. 13. 13. 13. 13. 13. 13. 13. 13. 13. 13. 13. 13. 13. 13. 13. 13.
  13. 13. 13. 13. 13. 13. 13.]
 [14. 14. 14. 14. 14. 14. 14. 14. 14. 14. 14. 14. 14. 14. 14. 14. 14. 14.
  14. 14. 14. 14. 14. 14. 14.]
 [15. 15. 15. 15. 15. 15. 15. 15. 15. 15. 15. 15. 15. 15. 15. 15. 15. 15.
  15. 15. 15. 15. 15. 15. 15.]
 [16. 16. 16. 16. 16. 16. 16. 16. 16. 16. 16. 16. 16. 16. 16. 16. 16. 16.
  16. 16. 16. 16. 16. 16. 16.]
 [17. 17. 17. 17. 17. 17. 17. 17. 17. 17. 17. 17. 17. 17. 17. 17. 17. 17.
  17. 17. 17. 17. 17. 17. 17.]
 [18. 18. 18. 18. 18. 18. 18. 18. 18. 18. 18. 18. 18. 18. 18. 18. 18. 18.
  18. 18. 18. 18. 18. 18. 18.]
 [19. 19. 19. 19. 19. 19. 19. 19. 19. 19. 19. 19. 19. 19. 19. 19. 19. 19.
  19. 19. 19. 19. 19. 19. 19.]]
The shape of pints lying on 2D Cartesian Grid:(500, 2)
```

[173]:
```python
#Computation of distance matrix D.

num_of_points = n1*n2 #Number of points in the grid.

#Initialize the distance matrix D.
D = np.zeros((num_of_points, num_of_points))
print(D.shape)
print(D)
```

```
#Compute the squared Euclidean distance matrix
for i in range(num_of_points):
    for j in range(num_of_points):
        D[i,j] = np.sum((X[i] - X[j])**2)

print("First 5*5 block of the distance matrix D: \n", D[:5,:5])
```

```
(500, 500)
[[0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 …
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]]
First 5*5 block of the distance matrix D:
 [[ 0.  1.  4.  9. 16.]
 [ 1.  0.  1.  4.  9.]
 [ 4.  1.  0.  1.  4.]
 [ 9.  4.  1.  0.  1.]
 [16.  9.  4.  1.  0.]]
```

2. In addition, create an array $c \in \mathbb{R}^{(n1 \cdot n2) \times 3}$, such that c[i,:] $\in \mathbb{R}^3$ is an RGB color code that encodes the position of point X[i,:] $\in \mathbb{R}^2$ in the grid. The exact choice of the encoding is up to you.

Answer: We can do the following steps to encode the position of X[i,:] $\in \mathbb{R}^2$ in the grid to c $\in \mathbb{R}^3$ as an RGB color code:

- Normalize the x and y coordinates to x and y to [0, 1] range.
- Use normalized x and y to RGB values.

[174]:
```
#Normalize the x and y coordinates of points on 2D cartesian grid.
x_norm = (X[:, 0] - X[:, 0].min()) / (X[:,0].max() - X[:,0].min())
y_norm = (X[:, 1] - X[:, 1].min()) / (X[:, 1].max() - X[:,1].min())
print(x_norm.shape)
print(y_norm)
```

```
(500,)
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.10526316 0.10526316 0.10526316 0.10526316
 0.10526316 0.10526316 0.10526316 0.10526316 0.10526316 0.10526316
 0.10526316 0.10526316 0.10526316 0.10526316 0.10526316 0.10526316
```

0.10526316 0.10526316 0.10526316 0.10526316 0.10526316 0.10526316
0.10526316 0.10526316 0.10526316 0.15789474 0.15789474 0.15789474
0.15789474 0.15789474 0.15789474 0.15789474 0.15789474 0.15789474
0.15789474 0.15789474 0.15789474 0.15789474 0.15789474 0.15789474
0.15789474 0.15789474 0.15789474 0.15789474 0.15789474 0.15789474
0.15789474 0.15789474 0.15789474 0.15789474 0.21052632 0.21052632
0.21052632 0.21052632 0.21052632 0.21052632 0.21052632 0.21052632
0.21052632 0.21052632 0.21052632 0.21052632 0.21052632 0.21052632
0.21052632 0.21052632 0.21052632 0.21052632 0.21052632 0.21052632
0.21052632 0.21052632 0.21052632 0.21052632 0.21052632 0.26315789
0.26315789 0.26315789 0.26315789 0.26315789 0.26315789 0.26315789
0.26315789 0.26315789 0.26315789 0.26315789 0.26315789 0.26315789
0.26315789 0.26315789 0.26315789 0.26315789 0.26315789 0.26315789
0.26315789 0.26315789 0.26315789 0.26315789 0.26315789 0.26315789
0.31578947 0.31578947 0.31578947 0.31578947 0.31578947 0.31578947
0.31578947 0.31578947 0.31578947 0.31578947 0.31578947 0.31578947
0.31578947 0.31578947 0.31578947 0.31578947 0.31578947 0.31578947
0.31578947 0.31578947 0.31578947 0.31578947 0.31578947 0.31578947
0.31578947 0.36842105 0.36842105 0.36842105 0.36842105 0.36842105
0.36842105 0.36842105 0.36842105 0.36842105 0.36842105 0.36842105
0.36842105 0.36842105 0.36842105 0.36842105 0.36842105 0.36842105
0.36842105 0.36842105 0.36842105 0.36842105 0.36842105 0.36842105
0.36842105 0.36842105 0.42105263 0.42105263 0.42105263 0.42105263
0.42105263 0.42105263 0.42105263 0.42105263 0.42105263 0.42105263
0.42105263 0.42105263 0.42105263 0.42105263 0.42105263 0.42105263
0.42105263 0.42105263 0.42105263 0.42105263 0.42105263 0.42105263
0.42105263 0.42105263 0.42105263 0.47368421 0.47368421 0.47368421
0.47368421 0.47368421 0.47368421 0.47368421 0.47368421 0.47368421
0.47368421 0.47368421 0.47368421 0.47368421 0.47368421 0.47368421
0.47368421 0.47368421 0.47368421 0.47368421 0.47368421 0.47368421
0.47368421 0.47368421 0.47368421 0.47368421 0.52631579 0.52631579
0.52631579 0.52631579 0.52631579 0.52631579 0.52631579 0.52631579
0.52631579 0.52631579 0.52631579 0.52631579 0.52631579 0.52631579
0.52631579 0.52631579 0.52631579 0.52631579 0.52631579 0.52631579
0.52631579 0.52631579 0.52631579 0.52631579 0.52631579 0.57894737
0.57894737 0.57894737 0.57894737 0.57894737 0.57894737 0.57894737
0.57894737 0.57894737 0.57894737 0.57894737 0.57894737 0.57894737
0.57894737 0.57894737 0.57894737 0.57894737 0.57894737 0.57894737
0.57894737 0.57894737 0.57894737 0.57894737 0.57894737 0.57894737
0.63157895 0.63157895 0.63157895 0.63157895 0.63157895 0.63157895
0.63157895 0.63157895 0.63157895 0.63157895 0.63157895 0.63157895
0.63157895 0.63157895 0.63157895 0.63157895 0.63157895 0.63157895
0.63157895 0.63157895 0.63157895 0.63157895 0.63157895 0.63157895
0.63157895 0.68421053 0.68421053 0.68421053 0.68421053 0.68421053
0.68421053 0.68421053 0.68421053 0.68421053 0.68421053 0.68421053
0.68421053 0.68421053 0.68421053 0.68421053 0.68421053 0.68421053
0.68421053 0.68421053 0.68421053 0.68421053 0.68421053 0.68421053
0.68421053 0.68421053 0.73684211 0.73684211 0.73684211 0.73684211

```
0.73684211 0.73684211 0.73684211 0.73684211 0.73684211 0.73684211
0.73684211 0.73684211 0.73684211 0.73684211 0.73684211 0.73684211
0.73684211 0.73684211 0.73684211 0.73684211 0.73684211 0.73684211
0.73684211 0.73684211 0.73684211 0.78947368 0.78947368 0.78947368
0.78947368 0.78947368 0.78947368 0.78947368 0.78947368 0.78947368
0.78947368 0.78947368 0.78947368 0.78947368 0.78947368 0.78947368
0.78947368 0.78947368 0.78947368 0.78947368 0.78947368 0.78947368
0.78947368 0.78947368 0.78947368 0.78947368 0.84210526 0.84210526
0.84210526 0.84210526 0.84210526 0.84210526 0.84210526 0.84210526
0.84210526 0.84210526 0.84210526 0.84210526 0.84210526 0.84210526
0.84210526 0.84210526 0.84210526 0.84210526 0.84210526 0.84210526
0.84210526 0.84210526 0.84210526 0.84210526 0.84210526 0.89473684
0.89473684 0.89473684 0.89473684 0.89473684 0.89473684 0.89473684
0.89473684 0.89473684 0.89473684 0.89473684 0.89473684 0.89473684
0.89473684 0.89473684 0.89473684 0.89473684 0.89473684 0.89473684
0.89473684 0.89473684 0.89473684 0.89473684 0.89473684 0.89473684
0.94736842 0.94736842 0.94736842 0.94736842 0.94736842 0.94736842
0.94736842 0.94736842 0.94736842 0.94736842 0.94736842 0.94736842
0.94736842 0.94736842 0.94736842 0.94736842 0.94736842 0.94736842
0.94736842 0.94736842 0.94736842 0.94736842 0.94736842 0.94736842
0.94736842 1.         1.         1.         1.         1.
1.         1.         1.         1.         1.         1.
1.         1.         1.         1.         1.         1.
1.         1.         1.         1.         1.         1.
1.         1.        ]
```

[175]:
```
#Generation of RGB colors.
c = np.zeros((n1*n2, 3)) #Initializing 500 rows and 3 columns
c[:, 0] = x_norm #Mapping x_norm to firt column of c as red color.
c[:, 1] = y_norm #Mapping y_norm to second column of c as green color.
c[:, 2] = (x_norm + y_norm) / 2 #Mapping the average of x_norm and y_norm to␣
 ↪third column of c as blue color.
print(f"Frist 5 rows of c(RGB) values: {c[:5]}")
```

```
Frist 5 rows of c(RGB) values: [[0.         0.         0.        ]
 [0.04166667 0.         0.02083333]
 [0.08333333 0.         0.04166667]
 [0.125      0.         0.0625    ]
 [0.16666667 0.         0.08333333]]
```

3. Based on the squared distances and the length scale parameter ϵ =1, compute now the matrix A, defined as follows: $A_{i,j} = \exp(-D_{i,j}/\epsilon^2)$ for $i \neq j$, $A_{i,i} = 0$. Then from A generate the graph Laplacian L and its eigendecomposition, as in the lecture.

[176]:
```
#Caculating Affinity Matrix A.
epsilon = 1 #as we are dealing with unit Euclidean distances.
A = np.exp(-D/(epsilon**2))
np.fill_diagonal(A,0) #Ensure the diagonal is zero.
```

```
#Caculating the degree matrix for Graph Laplacian.
degree_matrix = np.diag(np.sum(A, axis =1)) #Summing the diagonal of the column␣
 ↪of degree matrix.

#Computing Graph Labpalcian
L = degree_matrix - A

#Performing the eigendecomposition
eigen_values,eigen_vector = eigh(L)
print("Eigenvalues:\n", eigen_values)
print("Eigenvectors:\n", eigen_vector)
print(eigen_values.shape)
print(eigen_vector.shape)
```

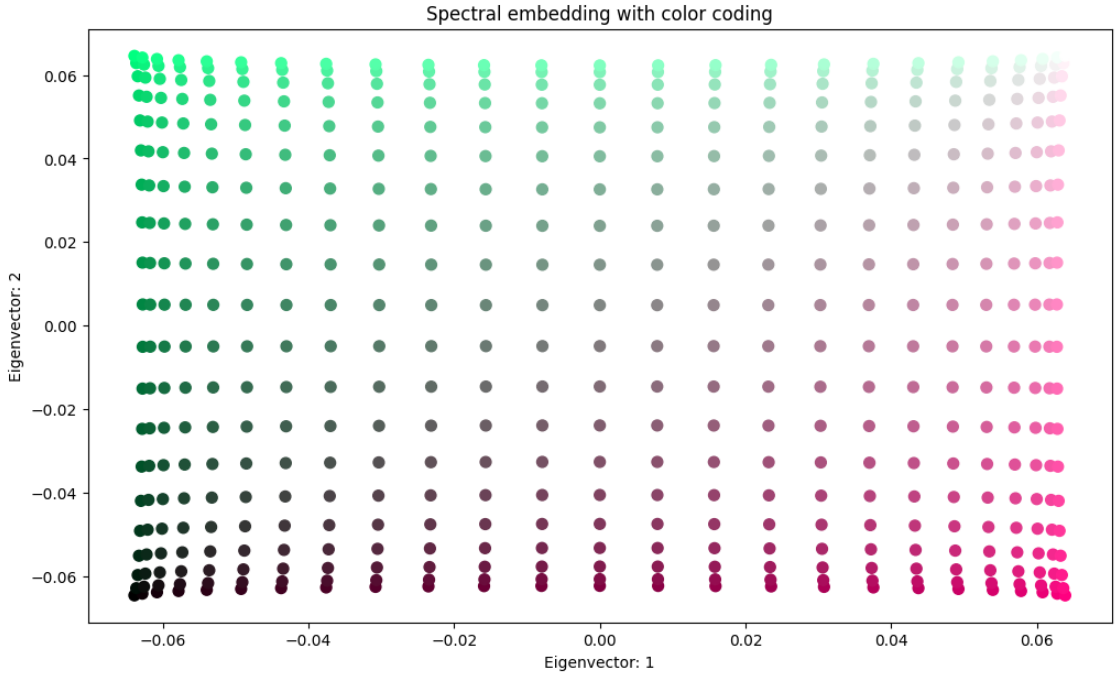
```
Eigenvalues:
 [0.         0.01206843 0.01891872 0.03028248 0.04793182 0.06510581
 0.07482244 0.08493252 0.10657121 0.11932506 0.12202513 0.1651582
 0.1732439  0.17549162 0.18629841 0.19941156 0.2072951  0.25219074
 0.26189542 0.28473256 0.28568308 0.29122311 0.29510896 0.32585827
 0.33645287 0.34772924 0.37832577 0.39902784 0.40630011 0.42909147
 0.43046795 0.43353668 0.44986845 0.4601874  0.47135327 0.52067795
 0.52541084 0.53013325 0.53837331 0.53905496 0.58720328 0.5886022
 0.59212791 0.59357421 0.63947361 0.64375154 0.66053065 0.66142738
 0.66266379 0.67316527 0.68481003 0.72730726 0.74810806 0.76169168
 0.76342198 0.76390475 0.77275804 0.79643016 0.79982343 0.80108643
 0.82562973 0.82727131 0.86557445 0.87733729 0.88443839 0.89066104
 0.91980509 0.92327766 0.9364732  0.93905907 0.9404066  0.94128822
 0.94405733 0.99641332 1.00674011 1.02330266 1.02485448 1.02883673
 1.03456139 1.05779592 1.07972443 1.08070087 1.08226165 1.09112382
 1.10967653 1.112737   1.11300582 1.13742586 1.13979587 1.17234963
 1.1760231  1.17990582 1.19572138 1.21597649 1.21638471 1.22658361
 1.24595684 1.25459439 1.25608371 1.26152486 1.27325902 1.2794066
 1.28020252 1.28614884 1.30186035 1.32001614 1.34276677 1.34578799
 1.34830701 1.35782955 1.36135617 1.38950675 1.39914778 1.40193374
 1.41270378 1.42957064 1.43009959 1.43710522 1.43717255 1.44069471
 1.45476809 1.4670236  1.47069447 1.47133911 1.49463877 1.51227876
 1.51754082 1.5178868  1.51990343 1.54388573 1.54702714 1.55340041
 1.55675249 1.57610945 1.58041042 1.58364326 1.58900142 1.6002763
 1.60989263 1.61666144 1.6274085  1.63828119 1.64884392 1.65105743
 1.65334265 1.67367942 1.68142323 1.68237547 1.68712769 1.68868756
 1.69193272 1.69443411 1.70758761 1.71043496 1.71179041 1.72886113
 1.75975285 1.76014091 1.7609423  1.76318162 1.77982982 1.78284692
 1.78383773 1.78430589 1.7924071  1.81545408 1.82246129 1.82273201
 1.82383962 1.82888915 1.83377098 1.84454396 1.84696421 1.85229545
 1.86286432 1.86578279 1.86624863 1.87564853 1.88055402 1.89257786
 1.89896177 1.902359   1.90935592 1.91574479 1.91891825 1.92052306
 1.93214057 1.93822922 1.93914086 1.95022182 1.95989133 1.96275029
```

1.98552152 1.98856194 1.9898971 1.99112904 1.99330989 1.99584921
1.99796234 1.99806832 2.00065406 2.0018983 2.00534391 2.00556379
2.02222088 2.02971964 2.03036151 2.03170094 2.04900133 2.0495994
2.05919948 2.05959204 2.06415346 2.06614815 2.06912148 2.08343725
2.08790945 2.08829469 2.09233122 2.09368713 2.09398298 2.10291901
2.1030173 2.10528975 2.1116735 2.11289055 2.11487791 2.11573775
2.11584116 2.12904448 2.12905602 2.1314702 2.13224057 2.13224585
2.14727698 2.14741868 2.14881411 2.15090209 2.15290831 2.16038414
2.18165735 2.19166239 2.19717592 2.19830205 2.19977592 2.2026277
2.20559716 2.2098429 2.21040895 2.21879392 2.22920416 2.22961693
2.24570702 2.25485587 2.26108048 2.26227785 2.26990762 2.28485541
2.2855986 2.28661446 2.29018704 2.29530585 2.2959747 2.30278849
2.30897546 2.3140032 2.31408822 2.31831901 2.32175579 2.33125442
2.33204338 2.33862955 2.34062998 2.3527629 2.35894014 2.36884623
2.37444576 2.37445399 2.38101226 2.38285636 2.38641867 2.38684777
2.39568893 2.40028722 2.40375617 2.40396433 2.40896443 2.40980367
2.41078438 2.41895277 2.4253443 2.43243985 2.4398408 2.44301677
2.4433971 2.44457815 2.44642973 2.45418345 2.45700728 2.4617002
2.46943439 2.47245974 2.47592149 2.47825045 2.47929485 2.48525217
2.48537311 2.48965888 2.49177899 2.4917995 2.50008428 2.5027387
2.50575424 2.51357386 2.51361424 2.51510518 2.52826615 2.52961432
2.53199979 2.53211755 2.53270979 2.53652028 2.53682695 2.53886099
2.54263309 2.54334708 2.5451833 2.54906039 2.55473083 2.5580625
2.55942267 2.56546399 2.56556138 2.5687991 2.57307439 2.57542277
2.57756015 2.57772384 2.57899552 2.58122452 2.59030799 2.59195972
2.59401367 2.59515777 2.59578474 2.59816914 2.59821721 2.59939013
2.60442538 2.60504032 2.60717206 2.60760254 2.60857781 2.61083541
2.6116044 2.61437913 2.61518581 2.61627464 2.61749613 2.62170505
2.62306757 2.62704654 2.63156334 2.63257368 2.63300497 2.63813971
2.64290337 2.6438747 2.65050287 2.65131502 2.65583952 2.655923
2.65728706 2.65827592 2.66058282 2.6625981 2.66326313 2.66473649
2.66537315 2.66770713 2.67262173 2.67672208 2.67749757 2.68405425
2.69154664 2.69354196 2.69383943 2.69625636 2.69739701 2.70255604
2.70561322 2.70795307 2.71155692 2.71180045 2.71269475 2.71617515
2.71814361 2.72403932 2.72427928 2.72804121 2.73156188 2.73283308
2.74105553 2.74369323 2.7471226 2.75110903 2.75128942 2.75270601
2.75553642 2.75598001 2.75786654 2.76514302 2.77071669 2.77116426
2.77412272 2.78210945 2.78238484 2.78496642 2.78546618 2.78747559
2.78775245 2.79636306 2.79693513 2.80580894 2.80839477 2.81012779
2.81194405 2.81515304 2.81689397 2.81928556 2.82496186 2.82503347
2.82672572 2.8349957 2.83968456 2.84166595 2.844047   2.84863459
2.85011364 2.85043048 2.85065001 2.86059002 2.86186943 2.86640236
2.86966562 2.87077084 2.87717213 2.88021549 2.88041017 2.88154066
2.88568271 2.89139364 2.8973285 2.8987852 2.90051875 2.90353129
2.90733648 2.90877045 2.91073463 2.91704353 2.92035529 2.92528368
2.92719323 2.93051712 2.93416759 2.9351066 2.93726683 2.93730848
2.94939607 2.94988584 2.9518567 2.95735523 2.95906827 2.96066805
2.96093371 2.96689631 2.97226229 2.97664633 2.97951778 2.98036093

```
 2.98133884 2.98713872 2.99184386 2.99213849 2.99830579 2.99874143
 3.00280057 3.00928898 3.00941448 3.0116655  3.01401647 3.02129298
 3.02353588 3.02603781 3.02710266 3.03469705 3.03474372 3.0399208
 3.04281436 3.04768186]
Eigenvectors:
 [[-4.47213595e-02 -6.38948359e-02 -6.45881400e-02 …  6.16760016e-05
   6.24495165e-05  3.18627139e-05]
 [-4.47213595e-02 -6.28071179e-02 -6.41776546e-02 …  2.27456250e-04
   2.31059421e-04  1.07769200e-04]
 [-4.47213595e-02 -6.07980762e-02 -6.38417080e-02 … -6.55557308e-04
  -6.60629969e-04 -3.21283846e-04]
 …
 [-4.47213595e-02  6.07980762e-02  6.38417080e-02 … -6.55557308e-04
  -6.60629969e-04  3.21283846e-04]
 [-4.47213595e-02  6.28071179e-02  6.41776546e-02 …  2.27456250e-04
   2.31059421e-04 -1.07769200e-04]
 [-4.47213595e-02  6.38948359e-02  6.45881400e-02 …  6.16760016e-05
   6.24495165e-05 -3.18627139e-05]]
(500,)
(500, 500)
```

4. Let `eigvec` be the list of eigenvectors of L, sorted by increasing eigenvalue. Show a spectral embedding of X as a scatter plot of `eigvec[k1]` versus `eigvec[k2]`, $k_1 = 1$, $k_2 = 2$, and use the color codes c to encode the original grid structure. You should find that this indeed recovers approximately the original two-dimensional grid structure.

[177]:
```
#Spectral Embedding using the second and third eigen vectors.
k1, k2 = 1, 2
plt.figure(figsize = (12,7))
plt.scatter(eigen_vector[:, k1], eigen_vector[:, k2], c = c, s = 50)
plt.title("Spectral embedding with color coding")
plt.xlabel(f'Eigenvector: {k1}')
plt.ylabel(f'Eigenvector: {k2}')
plt.show()
```



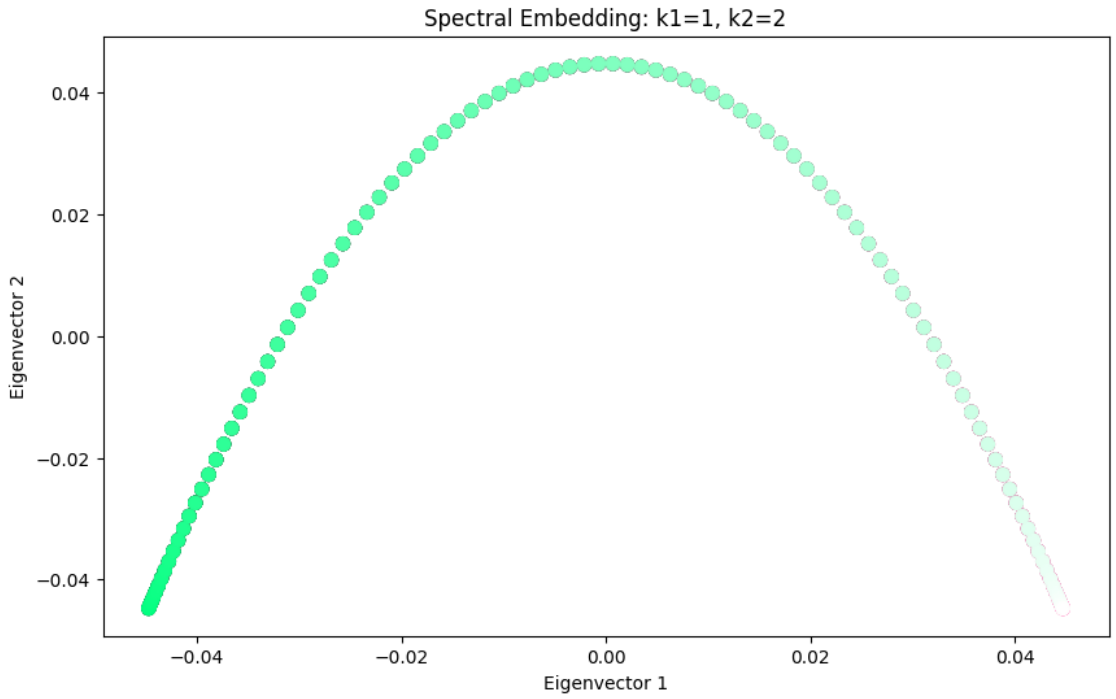
5. Now set n_1 =100, n_2 =10 and re-run the above code. $k_1 = 1$, $k_2 = 2$ now does no longer recover the two-dimensional grid structure. Find the best choices of k_1 and k_2 that do.

[178]:
```
#Generating example dataset.
n1, n2 = 100, 10
x = np.linspace(0, n1-1, n1)
y = np.linspace(0, n2-1, n2)
xx, yy = np.meshgrid(x, y)
X = np.vstack([xx.ravel(), yy.ravel()]).T

#Computing the matrix of squared Euclidean distances
num_of_points = n1*n2
D = np.zeros((num_of_points, num_of_points))

for i in range(num_of_points):
    for j in range(num_of_points):
        D[i,j] = np.sum((X[i] - X[j])**2)

#Compute the Affinity Matrix A
A = np.exp(-D / (epsilon**2))
np.fill_diagonal(A, 0)

#Compute the graph Laplacian L
Degree_matrix = np.diag(A.sum(axis =1))
L = Degree_matrix - A
```

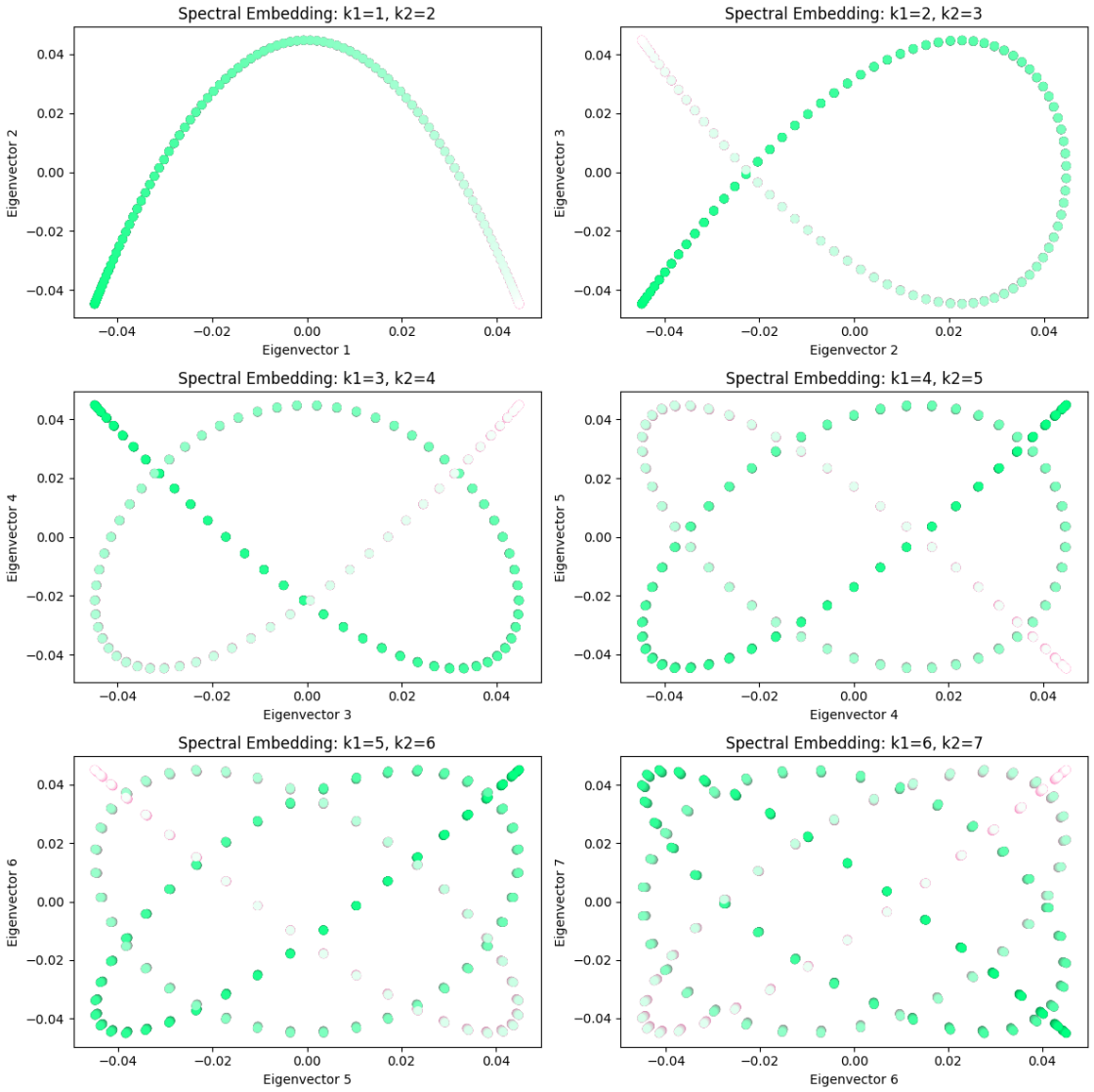
```
#Compute the eigendecomposition of Laplacian
eigenval, eigenvec = eigh(L)

#Create the color encoding.
c = np.zeros((n1 * n2, 3))

x_norm = (X[:, 0] - X[:, 0].min()) / (X[:, 0].max() - X[:,0].min())
y_norm = (X[:, 1] - X[:, 1].min()) / (X[:, 1].max() - X[:,1].min())

c[:, 0] = x_norm
c[:, 1] = y_norm
c[:, 2] = (x_norm + y_norm) / 2
```

[206]:
```
k1, k2 = 1,2
plt.figure(figsize=(10,6))
plt.scatter(eigenvec[:, k1], eigenvec[:, k2], c =c, s =50)
plt.title(f"Spectral Embedding: k1={k1}, k2={k2}")
plt.xlabel(f"Eigenvector {k1}")
plt.ylabel(f"Eigenvector {k2}")
```

[206]:
```
Text(0, 0.5, 'Eigenvector 2')
```



[204]:
```
# Visualize spectral embedding for different k1 and k2
plt.figure(figsize=(12, 12))
for i, (k1, k2) in enumerate([(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7)]):
    embedding = eigenvec[:, [k1, k2]]
    plt.subplot(3, 2, i + 1)
    plt.scatter(embedding[:, 0], embedding[:, 1], c =c)
    plt.title(f"Spectral Embedding: k1={k1}, k2={k2}")
    plt.xlabel(f"Eigenvector {k1}")
    plt.ylabel(f"Eigenvector {k2}")
plt.tight_layout()
plt.show()
```



Looking at the different graphs that are represented above, we can say that the best value of k_1

and k_2 that recover the two-dimensional grid structure is 5 and 6 respectively.

1.2 Exercise 8.2: Visualizing a relational database as decorated graph.

Consider a simple relational database that represents an online newspaper. *Journalists* can author *articles* (for simplicity, each article will be written by precisely one author) and articles can be *assigned* to (multiple) *categories*. *Readers* can create accounts and write *comments* on articles, they can express *reactions* to other readers' comments (such as 'agree' or 'disagree'), and they can *follow* certain authors (to be automatically informed, when they publish a new article). Assume that each of the concepts above that were highlighted in *italics* is represented by a separate table, and that relations between the concepts is encoded by simple key/foreign key references.

1. For each of the above tables, except for the *follow* and *assignment* tables, list at least two examples of columns that these tables should have (beyond keys and foreign keys).

[211]:
```python
def print_table_schema(table_name, columns):
    print(f"Table: {table_name}")
    print("-" * 50)
    print("| Column Name     | Description                    |")
    print("-" * 50)
    for column_name, description in columns.items():
        print(f"| {column_name:<15} | {description:<30} |")
    print("-" * 50)
    print()

# Define the schema for each table
schema = {
    'Journalists': {
        'journalist_id': 'Primary key',
        'name': 'Name of the journalist',
        'email': 'Email address of the journalist',
        'bio': 'Biography or description',
    },
    'Articles': {
        'article_id': 'Primary key',
        'journalist_id': 'Foreign Key referencing author',
        'title': 'Title of the article',
        'content': 'Full text or summary',
    },
    'Categories': {
        'category_id': 'Primary key',
        'article_id': 'Foreign Key referencing the article',
        'name': 'Name of the category',
        'description': 'Description of the category'
    },
    'Readers': {
        'reader_id': 'Primary key',
        'article_id': 'Foreign Key referencing article',
```

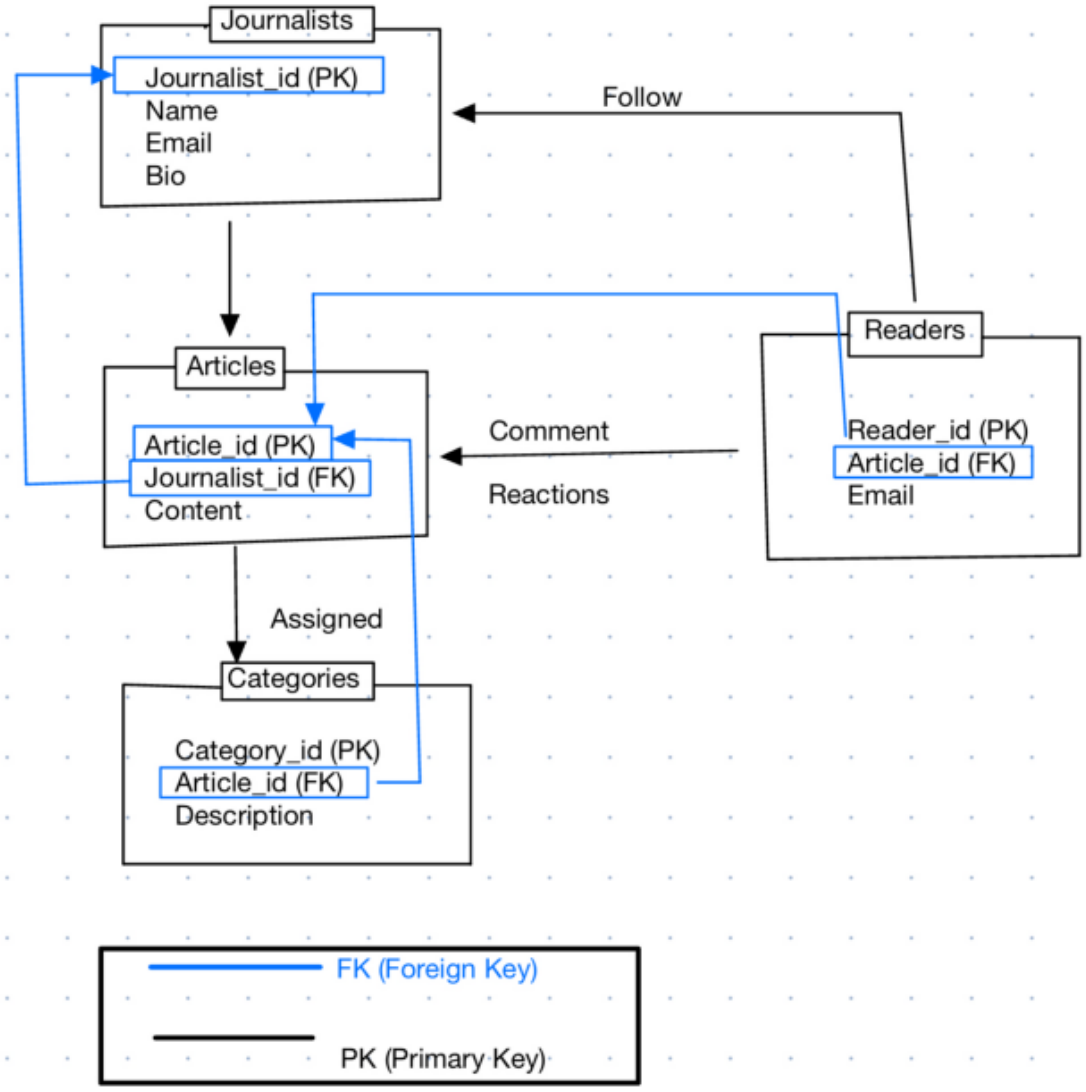
```
        'email': 'Email address',
    },
}

# Print each table schema
for table_name, columns in schema.items():
    print_table_schema(table_name, columns)
```

```
Table: Journalists
--------------------------------------------------
| Column Name     | Description                    |
--------------------------------------------------
| journalist_id   | Primary key                    |
| name            | Name of the journalist         |
| email           | Email address of the journalist |
| bio             | Biography or description       |
--------------------------------------------------

Table: Articles
--------------------------------------------------
| Column Name     | Description                    |
--------------------------------------------------
| article_id      | Primary key                    |
| journalist_id   | Foreign Key referencing author |
| title           | Title of the article           |
| content         | Full text or summary           |
--------------------------------------------------

Table: Categories
--------------------------------------------------
| Column Name     | Description                    |
--------------------------------------------------
| category_id     | Primary key                    |
| article_id      | Foreign Key referencing the article |
| name            | Name of the category           |
| description     | Description of the category    |
--------------------------------------------------

Table: Readers
--------------------------------------------------
| Column Name     | Description                    |
--------------------------------------------------
| reader_id       | Primary key                    |
| article_id      | Foreign Key referencing article |
| email           | Email address                  |
--------------------------------------------------
```

2. Draw a graph that represents the above database, in particular the table columns and reference relations. Possibly you can choose a separate visual representation for the relations encoded by the auxiliary tables *follow* and *assigned*. You can do this in any software you want, or with a simple hand drawing (scanned or on a tablet).

[228]:
```
image_path = 'Graph_representation_database.jpeg'
image = Image.open(image_path)
plt.figure(figsize=  (12,8))
plt.imshow(image)
plt.axis('off')
```

[228]:
```
(-0.5, 1356.5, 1371.5, -0.5)
```



[ ]:

[ ]: