1. *Order the faces:* The following dataset contains 33 faces of the same person ($Y \in \mathbb{R}^{112\times 92\times 33}$) in different angles,

   `https://yao-lab.github.io/data/face.mat`

   You may create a data matrix $X \in \mathbb{R}^{n\times p}$ where $n = 33, p = 112 \times 92 = 10304$ (e.g. `X=reshape(Y,[10304,33])';` in matlab).

   (a) Explore the MDS-embedding of the 33 faces on top two eigenvectors: order the faces according to the top 1st eigenvector and visualize your results with figures.

   (b) Explore the ISOMAP-embedding of the 33 faces on the $k = 5$ nearest neighbor graph and compare it against the MDS results. Note: you may try Tenenbaum's Matlab code `https://yao-lab.github.io/data/isomapII.m`

   (c) Explore the LLE-embedding of the 33 faces on the $k = 5$ nearest neighbor graph and compare it against ISOMAP. Note: you may try the following Matlab code `https://yao-lab.github.io/data/lle.m`

2. *Manifold Learning*: The following codes by Todd Wittman contain major manifold learning algorithms talked on class.

   `http://math.stanford.edu/~yuany/course/data/mani.m`

   Precisely, eight algorithms are implemented in the codes: MDS, PCA, ISOMAP, LLE, Hessian Eigenmap, Laplacian Eigenmap, Diffusion Map, and LTSA. The following nine examples are given to compare these methods,

   (a) Swiss roll;

   (b) Swiss hole;

   (c) Corner Planes;

   (d) Punctured Sphere;

   (e) Twin Peaks;

   (f) 3D Clusters;

   (g) Toroidal Helix;

   (h) Gaussian;

   (i) Occluded Disks.

The solution of problem 1 below is the Face Manifold.pdf

The solution of problem 2 below will be in the Comparison.pdf.

FaceManifold

April 10, 2021

[63]:
```
from time import time
import numpy as np
import numpy.linalg as alg
import pandas as pd

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.ticker as mtick
from matplotlib import offsetbox
from sklearn.manifold import MDS, Isomap, LocallyLinearEmbedding
```

[44]:
```
# Load Data
X = np.loadtxt('face.csv', delimiter=',')
X = X.T

# Identify the shape
(N, P)= X.shape
print('The shape of face matrix now is {}'.format(X.shape))

# Normalization
one = np.ones((N,1))
X = X-one.dot(one.T).dot(X)
```

```
(10304, 33)
The shape of face matrix now is (33, 10304)
```

[45]:
```
# SVD by nimpy
# u @ np.diag(s) @ vh = (u * s) @ vh
u, s, vh = alg.svd(X)

print('The shape of u is', u.shape)
print('The shape of s is', s.shape)
print('The shape of v is', vh.shape)

M = u[:,:2]
p1 = M[:, 0]
p2 = M[:, 1]
```

```
pairs = [(p1[i], i) for i in range(N)]
pairs.sort(key = lambda pairs: pairs[0])

order = [j for (i, j) in pairs]
```

```
The shape of u is (33, 33)
The shape of s is (33,)
The shape of v is (10303, 10303)
```

[71]:
```
plt.figure()
cmap = cm.gray_r
# Attention! Load X again for the natural arangement of the photo matrix is␣
 ↪broken
# when the tensor is reduced from 3 dimension to 2 dimension.
X = np.loadtxt('face.csv', delimiter=',')

for i in range(N):
    idx = order[i]
    pic = X[:, idx]
    pic_matrix = np.reshape(pic, (92, 112))

    plt.subplot(3, 11, i+1)
    plt.imshow(pic_matrix.T, cmap=cmap)
    plt.xticks([])
    plt.yticks([])

plt.show()
```

[72]:
```
Original_X = X
X = X.T
print(X.shape)
```

```
(33, 10304)
```

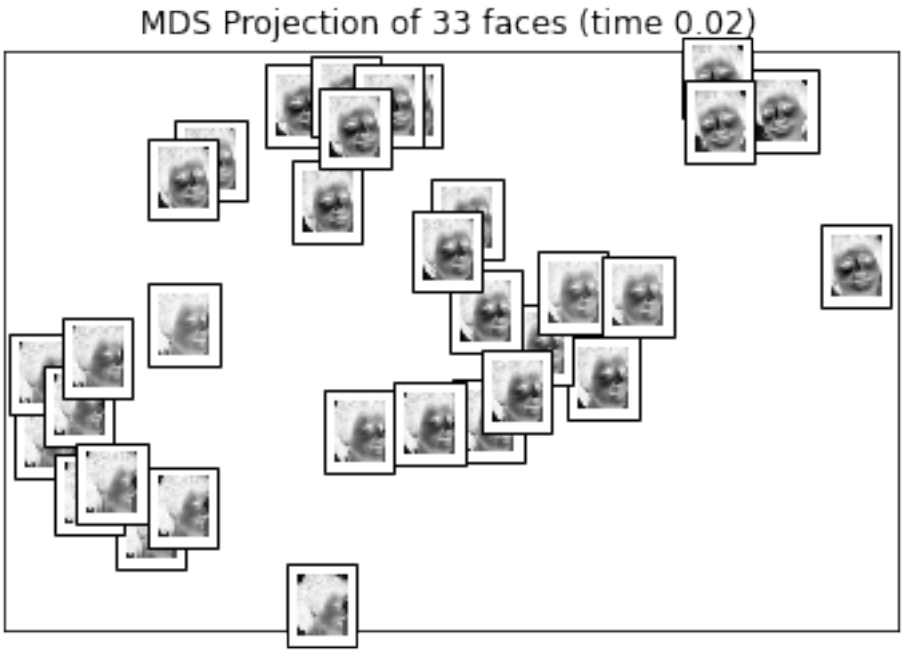
[77]:
```
def plot_embedding(X, title=None):
    x_min, x_max = np.min(X, 0), np.max(X, 0)
    X = (X - x_min) / (x_max - x_min)

    plt.figure()
    ax = plt.subplot(111)
    plt.scatter(X[:, 0], X[:, 1])

    if hasattr(offsetbox, 'AnnotationBbox'):
        for i in range(X.shape[0]):
            pic = Original_X[:, i]
            pic_matrix = np.reshape(pic, (92, 112))
            imagebox = offsetbox.AnnotationBbox(
                offsetbox.OffsetImage(pic_matrix.T, cmap = cmap, zoom=.2),
                X[i])
            ax.add_artist(imagebox)
    plt.xticks([])
    plt.yticks([])
    if title is not None:
        plt.title(title)
    pic_matrix = np.reshape(pic, (92, 112))
    plt.show()
```

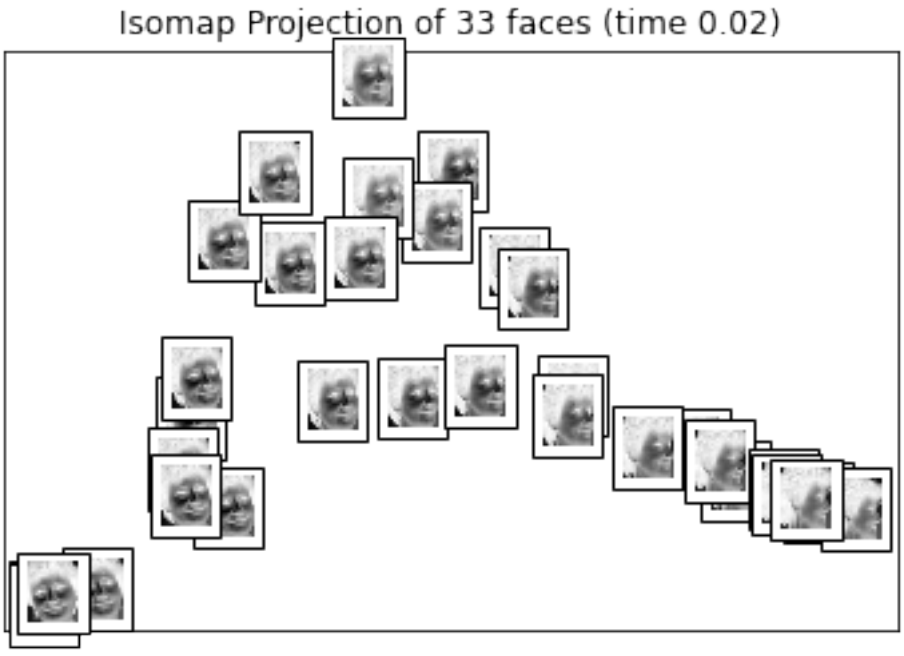
[78]:
```
print('MDS Projection of 33 faces')
t0 = time()
mds = MDS(n_components=2, n_init=1, max_iter=100)
X_mds = mds.fit_transform(X)
plot_embedding(X_mds, 'MDS Projection of 33 faces (time %.2f)'%(time()-t0))
```

```
MDS Projection of 33 faces
```



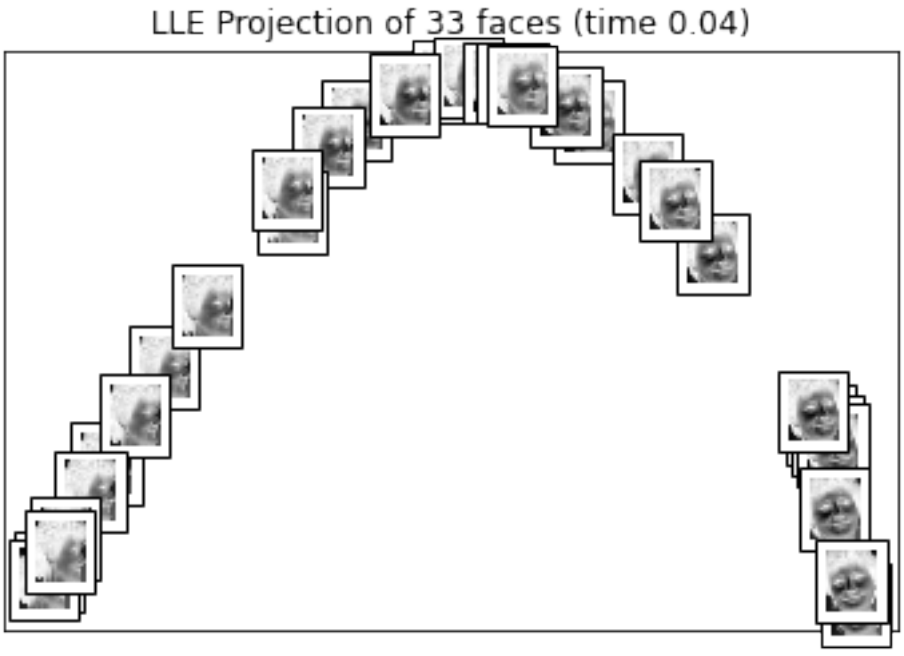
[79]:
```
print("Computing Isomap Projection of 33 faces")
t0 = time()
iso = Isomap(n_neighbors=5, n_components=2)
X_iso = iso.fit_transform(X)
plot_embedding(X_iso, 'Isomap Projection of 33 faces (time %.2f)'%(time()-t0))
```

```
Computing Isomap Projection of 33 faces
```



[80]:
```
print('Computing LLE of 33 faces')
t0 = time()
lle = LocallyLinearEmbedding(n_neighbors=5, n_components=2)
X_lle = lle.fit_transform(X)
plot_embedding(X_lle, 'LLE Projection of 33 faces (time %.2f)'%(time()-t0))
```

```
Computing LLE of 33 faces
```


LLE Projection of 33 faces (time 0.04)

[ ]:

Manifold Learning Comparison

Introduction

My comparison between different methods is based on the following aspects:

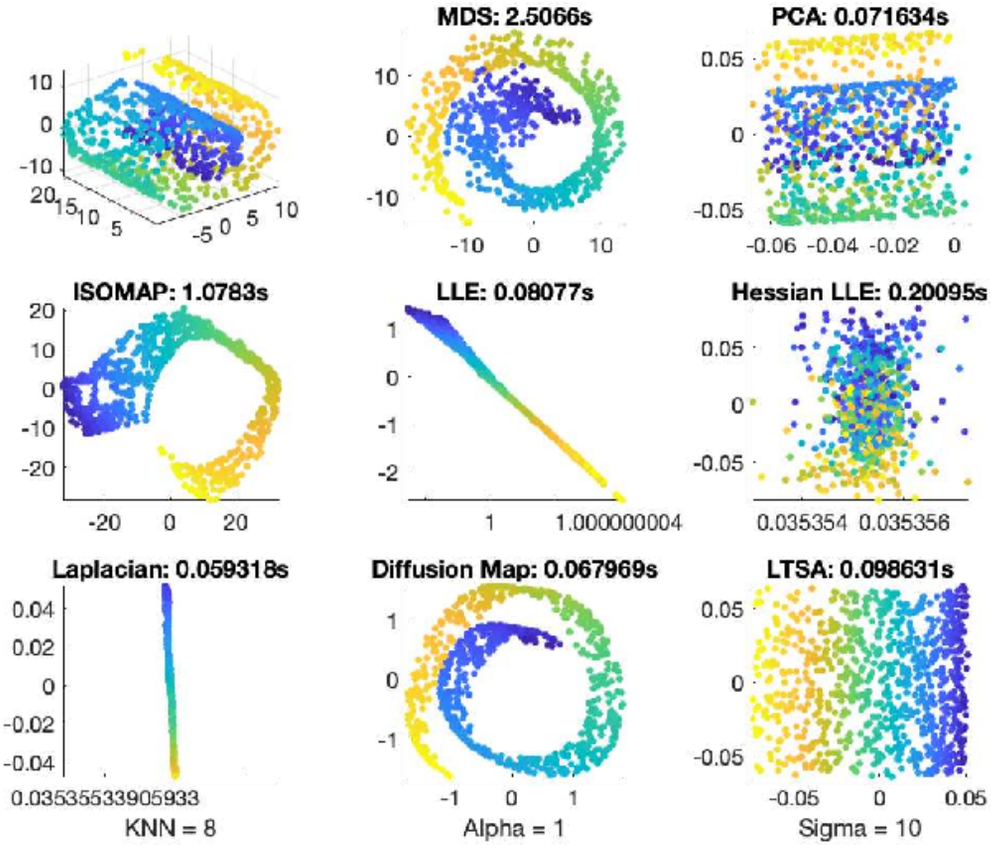
- Time Used
- Sensitivity to the parameters throughout

Other possible considerations are:

- Non-convexity
- Manifold Geometry
- Sparse Data
- Curvature
- Non-uniform Sampling
- Clustering
- Corners
- Noise

Experiments

Swiss Roll

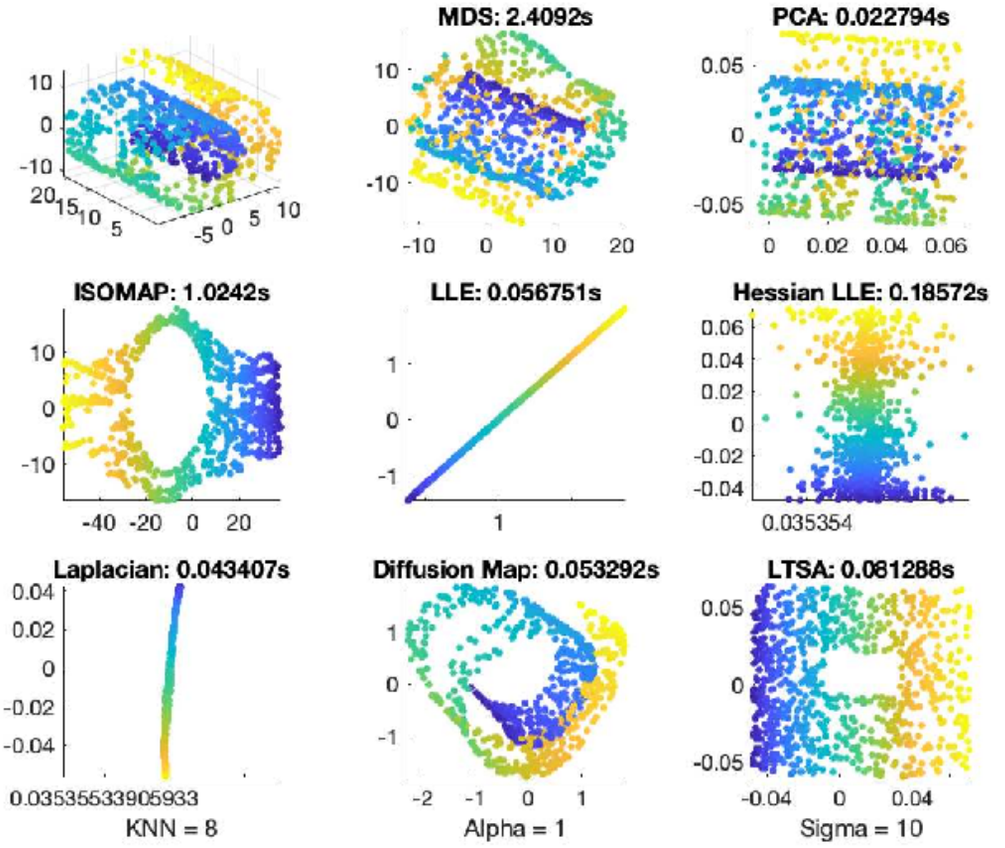


We see that

1. Slow methods are MDS>Isomap>Hessian LLE
2. MDS and PCA cannot unfold the Swiss Roll, no manifold info is used
3. Laplacian cannot handle this data properly
4. Diffusion Maps and Swiss Roll cannot unfold the Swiss Roll either

Swiss Hole (Non-Convexity)

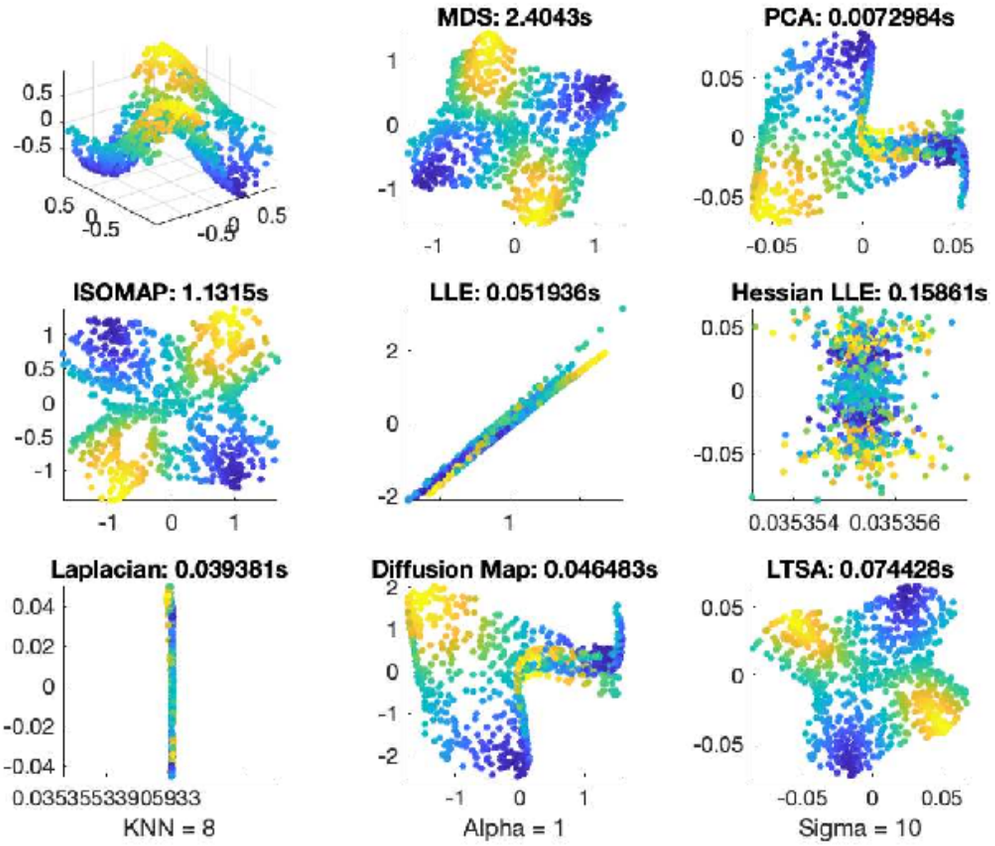
Can the Swiss Roll be unfolded if there exists a hole?



We see clearly that:

1. Only LTSA handles the non-convex Swiss Hole successfully
2. Isomap gets a flat swiss hole with distortion
3. MDS, PCA and Diffusion Map fails to unfold the Roll
4. LLE and Laplacian fails totally

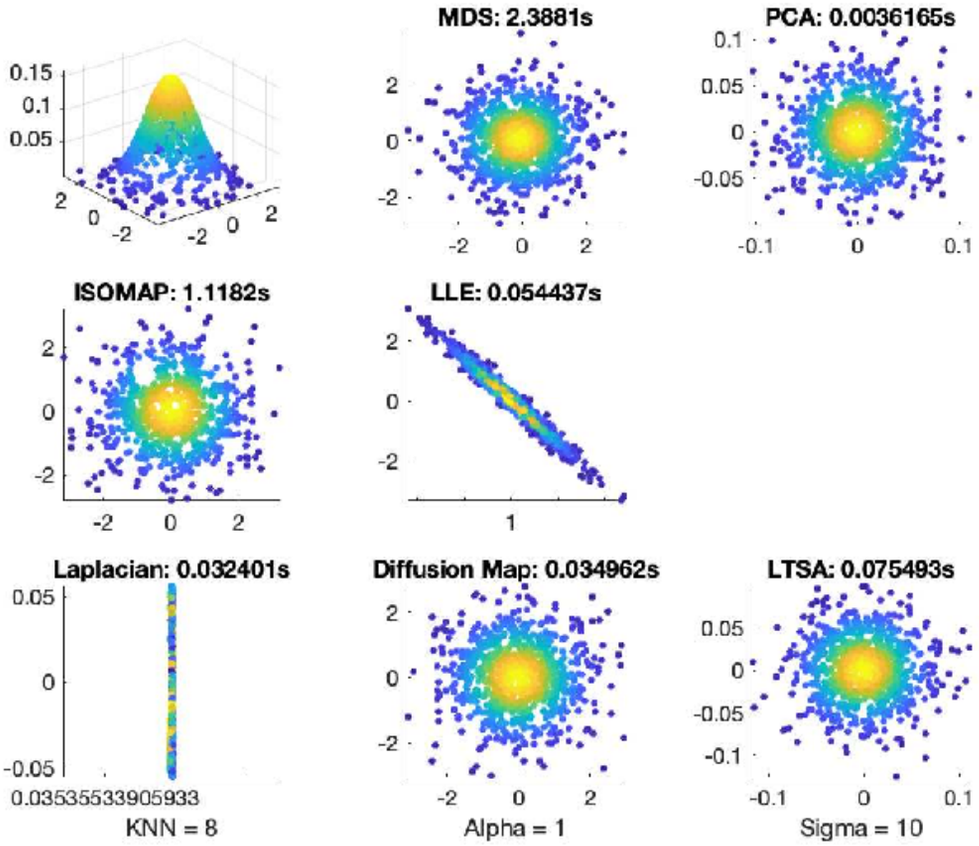
Twin Peaks (Curvature)



We see that:

1. LLE, Hessian LLE and Laplacian fails
2. All other method successfully unfold with distortion to different degree
3. The best result is given by MDS

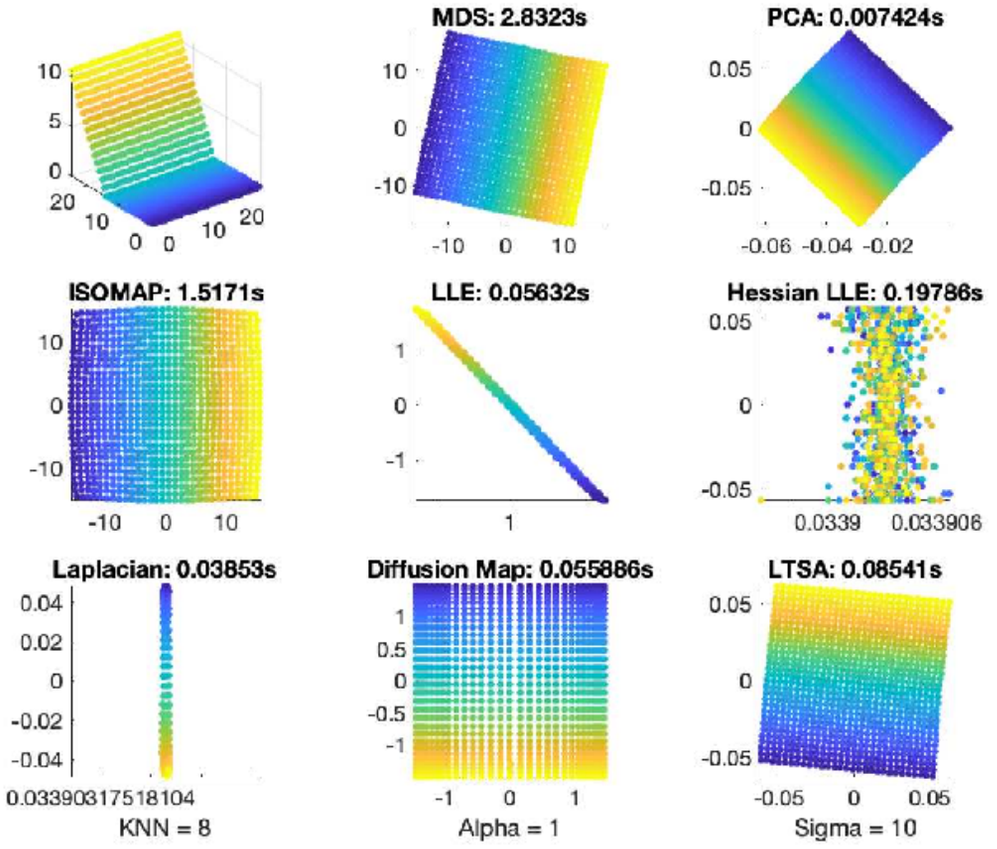
Gaussian (Non-uniform Sampling and Curvature)



We see that:

1. Simple methods like MDS, PCA, Isomap, Diffusion Map and LTSA perform perfectly
2. We see LLE and Laplacian fails
3. We see poor Hessian LLE breaks down

Corners Planes

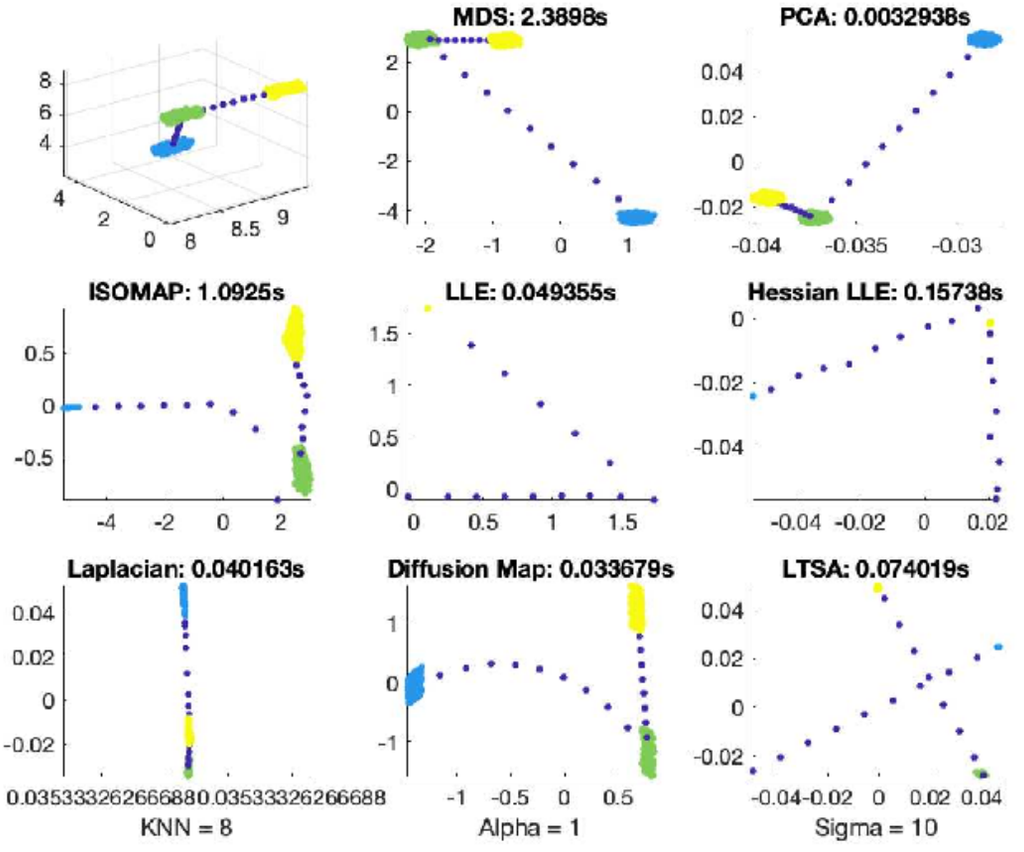


We see that:

1. The performance is similar to the case of Gaussian.
2. Simple methods like MDS, PCA, Isomap, Diffusion Map and LTSA perform perfectly
3. We see LLE and Laplacian fails
4. We see a Hessian LLE with chaos

3D-Cluster(Cluster and Sparsity)

A good mapping should preserve the clustered data
We generate three non-overlapping clusters with random centers and then connect the clusters with a line.

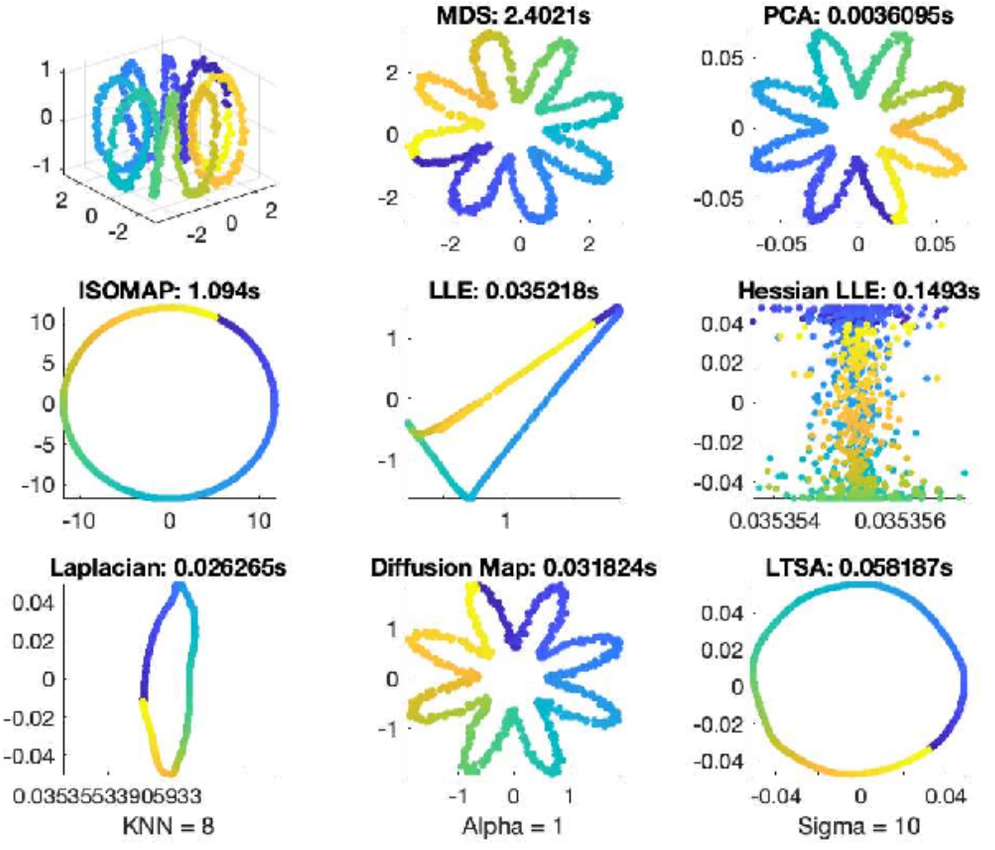


We see that:

1. MDS and PCA works well
2. Isomap and Diffusion Map restore the manifold structure with a little bit distortion
3. LLE, Hessian LLE and LTSA compressed each cluster into a single point
4. Laplacian has overlapped sparse connecting lines.
5. LTSA has crossed sparse connecting lines.

Toroidal Helix (Sparsity)

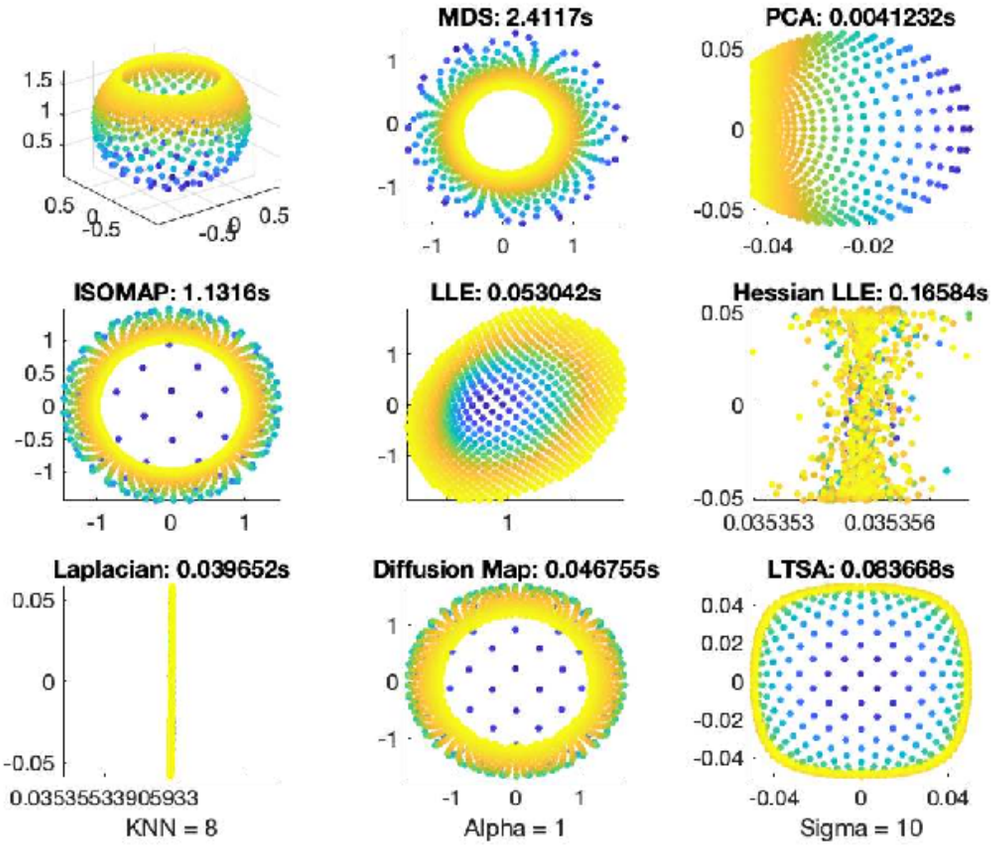
Can the method handle changes from dense to sparse regions?



We see that:

1. Isomap and LSTA are correct
2. LLE AND Laplacian also showed the part of the loop structure, with great distortion though
3. MDS, PCA and Diffusion Maps shows an asterisk
4. Hessian LLE fails

Punctured Sphere (Sparsity)

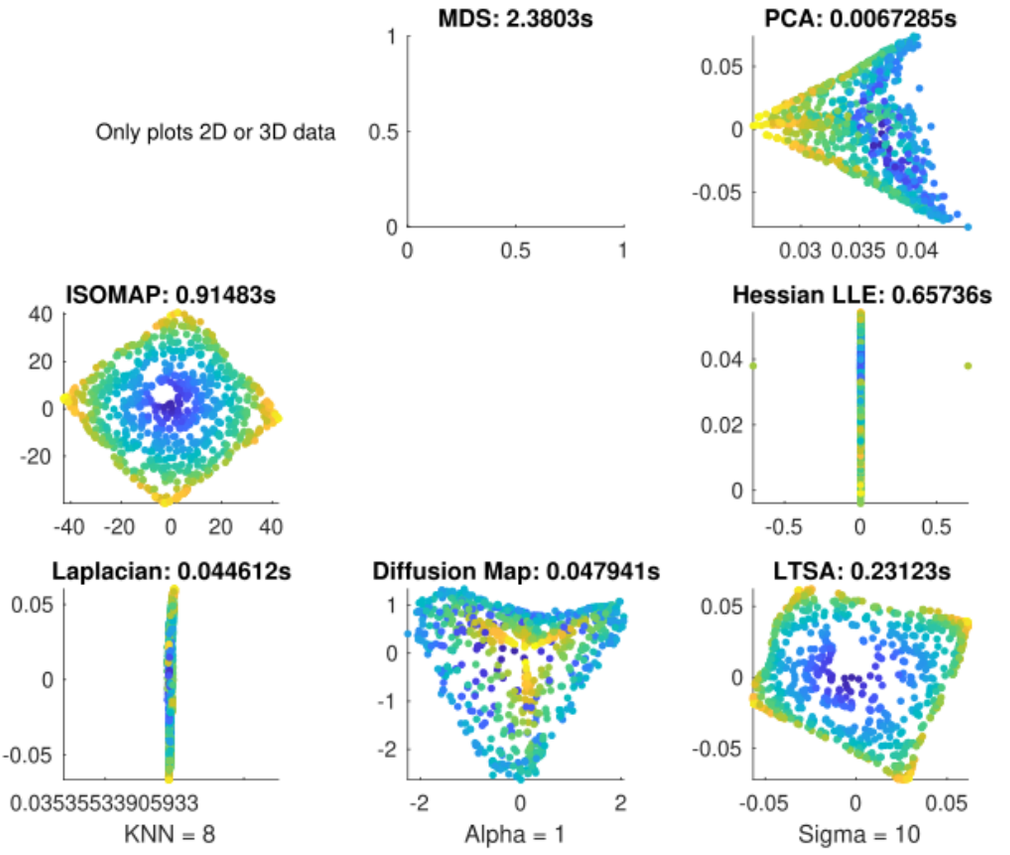


We see that:

1. LLE is right
2. Isomap and Diffusion Maps are also correct but emphasize too much on the sparsity
3. LTSA is kind of correct with a distorted square shape
4. PCA projects the sphere to one side while MDS turns the outside into the heart
5. Laplacian and Hessian LLE fail

Occluded Disk(High-Dimensional Data)

We Create 20*20 images with a disk of fixed radius and random center.



We see that:

1. Some methods break down like MDS, LLE(crashed)
2. Isomap performs good
3. LTSA performs second good
4. Diffusion Map turns the outside into the heart
5. Laplacian and Hessian LLE fail

Summary

1: Handels successfully
0: Fails to handel

Handles?	MDS	PCA	ISOMAP	LLE	Hessian	Laplacian	Diffusion	LTSA
speed	0	1	1	1	0	1	1	1
geometry	0	0	1	1	1	1	Nah	Nah
non-convexity	0	0	0	Nah	1	Nah	Nah	Nah
non-uniform-sampling	1	1	1	1	Nah	0	1	1
curvature	0	0	1	Nah	1	1	1	1
corner	0	0	1	1	0	1	1	1
cluster	1	1	1	1	0	0	1	1
noise	1	1	Nah	0	1	1	1	1
sparsity	1	1	1	1	0	1	0	0
sensitivity	0	0	1	1	1	1	1	1

3. *Nyström method:* In class, we have shown that every manifold learning algorithm can be regarded as Kernel PCA on graphs: (1) given N data points, define a neighborhood graph with N nodes for data points; (2) construct a positive semidefinite kernel K; (3) pursue spectral decomposition of K to find the embedding (using top or bottom eigenvectors). However, this approach might suffer from the expensive computational cost in spectral decomposition of K if N is large and K is non-sparse, e.g. ISOMAP and MDS.

To overcome this hurdle, Nyström method leads us to a scalable approach to compute eigenvectors of low rank matrices. Suppose that an N-by-N positive semidefinite matrix $K \succeq 0$ admits the following block partition

$$K = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}. \tag{1}$$

where A is an n-by-n block. Assume that A has the spectral decomposition $A = U\Lambda U^T$, $\Lambda = \mathrm{diag}(\lambda_i)$ ($\lambda_1 \geq \lambda_2 \geq \ldots \lambda_k > \lambda_{k+1} = \ldots = 0$) and $U = [u_1, \ldots, u_n]$ satisfies $U^T U = I$.

(a) Assume that $K = XX^T$ for some $X = [X_1; X_2] \in \mathbb{R}^{N \times k}$ with the block $X_1 \in \mathbb{R}^{n \times k}$. Show that X_1 and X_2 can be decided by:

$$X_1 = U_k \Lambda_k^{1/2}, \tag{2}$$

$$X_2 = B^T U_k \Lambda_k^{-1/2}, \tag{3}$$

where $U_k = [u_1, \ldots, u_k]$ consists of those k columns of U corresponding to top k eigenvalues λ_i ($i = 1, \ldots, k$).

Since $A = U\Lambda U^T$ $\quad \Lambda = \mathrm{diag}(\lambda_i)$ $\quad (\lambda_1 \geq \lambda_2 \geq \cdots \lambda_k > \lambda_{k+1} = 0 \cdots = 0)$

Thus $\quad A = U_k \Lambda_k U_k^T$

where $\quad U_k = [u_1, \cdots u_k]$ consists of top k eigenvectors

$\Lambda_k = \mathrm{diag}(\lambda_1, \lambda_2, \cdots \lambda_k)$

$$X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$$

$$K = XX^T = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} (X_1^T; X_2^T) = \begin{pmatrix} X_1 X_1^T & X_1 X_2^T \\ X_2 X_1^T & X_2 X_2^T \end{pmatrix}$$

① $X_1 X_1^T = A = U_k \Lambda_k U_k^T$

Λ_k is a diagonal matrix whose square root is $\Lambda_k^{\frac{1}{2}}$

$$X_1 X_1^T = U_k \Lambda_k^{\frac{1}{2}} \Lambda_k^{\frac{1}{2}} U_k^T = U_k \Lambda_k^{\frac{1}{2}} (\Lambda_k^{\frac{1}{2}})^T U_k^T$$

$$= (U_k \Lambda_k^{\frac{1}{2}})(U_k \Lambda_k^{\frac{1}{2}})^T$$

Thus, if we assign $X_1 \triangleq U_k \Lambda_k^{\frac{1}{2}}$

we do have $X_1 X_1^T = A$

② $X_1 X_2^T = B$

$X_2^T = X_1^{-1} B = (\Lambda_k^{\frac{1}{2}})^{-1} \cdot U_k^{-1} B = \Lambda_k^{-\frac{1}{2}} U_k^T B$

$X_2 = B^T U_k \cdot \Lambda_k^{-\frac{1}{2}}$

Thus, we can identify $X_1 = U_k \Lambda_k^{1/2}$ and $X_2 = B^T U_k \Lambda_k^{-\frac{1}{2}}$

(b) Show that for general $K \succeq 0$, one can construct an approximation from (2) and (3),

$$\hat{K} = \begin{bmatrix} A & B \\ B^T & \hat{C} \end{bmatrix}. \tag{4}$$

where $A = X_1 X_1^T$, $B = X_1 X_2^T$, and $\hat{C} = X_2 X_2^T = B^T A^\dagger B$, $A^\dagger$ denoting the Moore-Penrose (pseudo-) inverse of A. Therefore $\|\hat{K} - K\|_F = \|C - B^T A^\dagger B\|_F$. Here the matrix $C - B^T A^\dagger B =: K/A$ is called the (generalized) *Schur Complement* of A in K.

Once we know $X_1 = U_k \Lambda_k^{1/2}$ and $X_2 = B^T U_k \Lambda_k^{-\frac{1}{2}}$

$$\hat{K} = X X^T = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \begin{pmatrix} X_1^T & X_2^T \end{pmatrix} = \begin{pmatrix} X_1 X_1^T & X_1 X_2^T \\ X_2 X_1^T & X_2 X_2^T \end{pmatrix}$$

$X_1 X_1^T = U_k \Lambda_k^{\frac{1}{2}} \Lambda_k^{\frac{1}{2}} U_k^T = U_k \Lambda_k U_k^T = A$

$X_1 X_2^T = U_k \Lambda_k^{\frac{1}{2}} \Lambda_k^{-\frac{1}{2}} U_k^T B = U_k U_k^T B = B$

$X_2 X_1^T = (X_1 X_2^T)^T = B^T$

$X_2 X_2^T = B^T U_k \Lambda_k^{-1} U_k^T B$

Assign. $G \triangleq U_k \Lambda_k^{-1} U_k^T$

we have:

(1) $GAG = (U_K \Lambda_K^{-1} U_K^T)(U_K \Lambda_K U_K^T)(U_K \Lambda_K^{-1} U_K^T)$

$= U_K \Lambda_K^{-1} U_K = G.$

(2) $AGA = (U_K \Lambda_K U_K^T)(U_K \Lambda_K^{-1} U_K^T)(U_K \Lambda_K U_K^T)$

$= U_K \Lambda_K U_K^T = A$

(3) $(AG)^T = (U_K \Lambda_K U_K^T U_K \Lambda_K^{-1} U_K^T)^T = I = AG$

(4) $(GA)^T = (U_K \Lambda_K^{-1} U_K^T U_K \Lambda_K U_K^T)^T = I = GA$

By definition, G is the Moore-Penrose inverse of A and it must be unique. $A^\dagger = G$

Therefore

$$\|K - \hat{K}\|_F = \sqrt{\|A - A\|_F^2 + \|B - B\|_F^2 + \|B^T - B^T\|_F^2 + \|C - B^T A^\dagger B\|_F^2}$$

(c) Explore Nyström method on the Swiss-Roll dataset (`http://yao-lab.github.io/data/swiss_roll_data.mat` contains 3D-data X; `http://yao-lab.github.io/data/swissroll.m` is the matlab code) with ISOMAP. To construct the block A, you may choose either of the following:

n random data points;