Scalable ML

10605-10805

Manifold Learning

Barnabás Póczos

Motivation

- **Issues**: Difficult to visualize data in dimensions greater than three.

- **Goal:** Find meaningful **low-dimensional structures** hidden in **high-dimensional** observations.

  The human brain confronts the same problem in perception:

  - 30,000 auditory nerve fibers
  - 10^6 optic nerve fibers

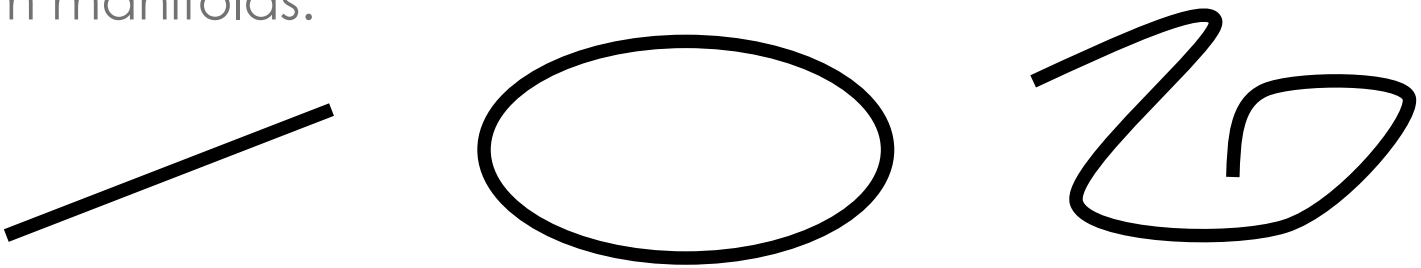
  Goal: extract small number of perceptually relevant features.

Nonlinear Embedding = Manifold Learning

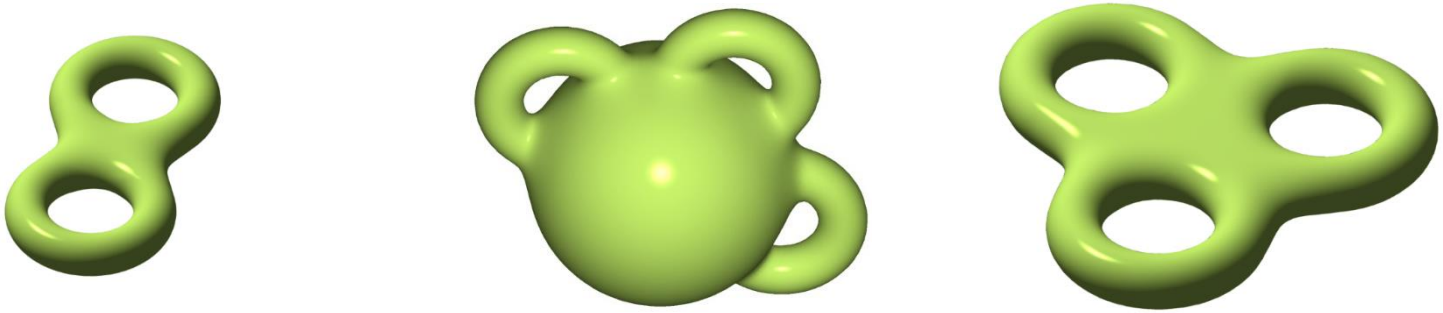
Informal definition:

Manifold = any object which is nearly "flat" on small scales.

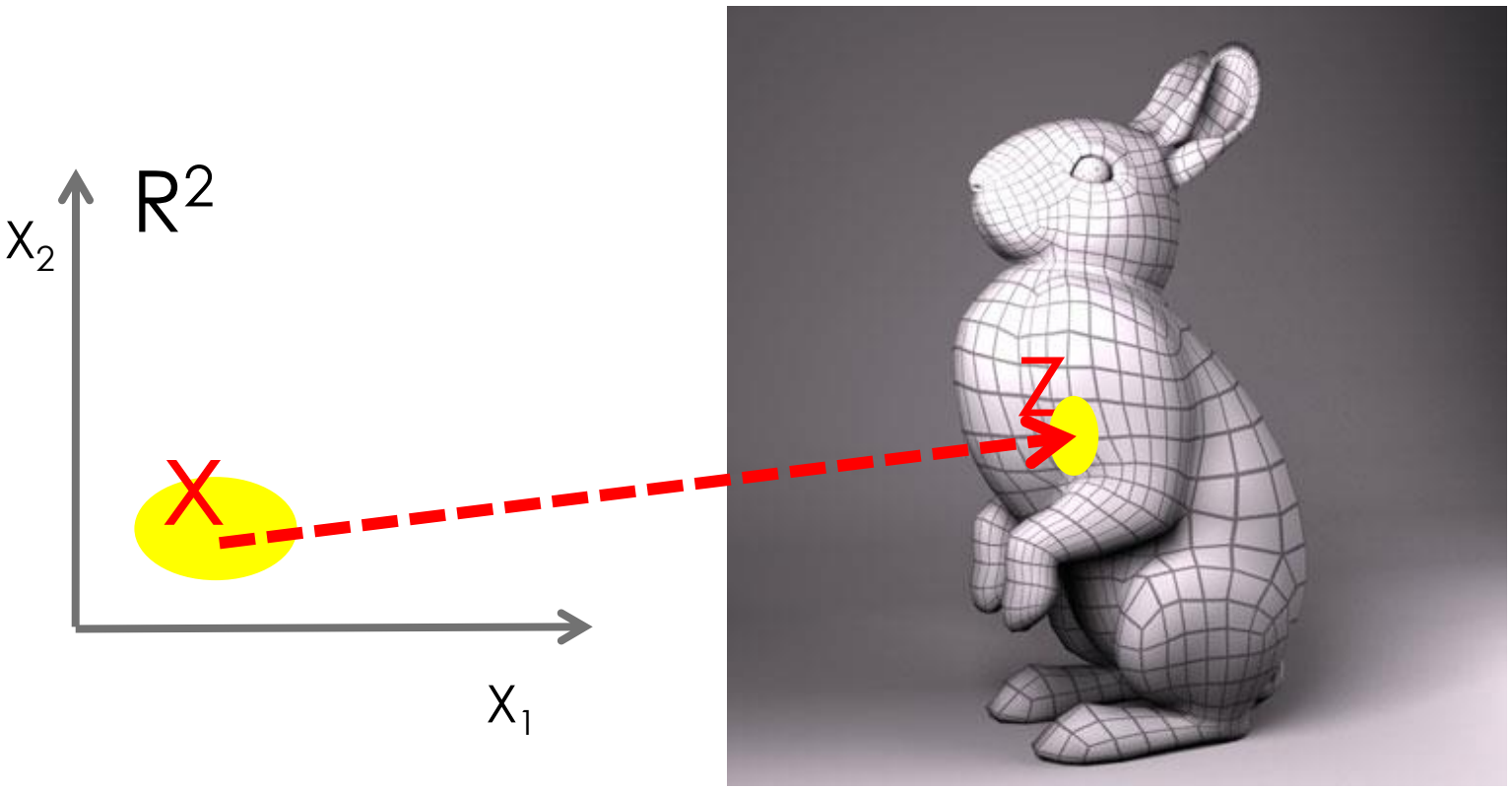
1dim manifolds:



2dim manifolds:



Manifold Learning



Manifold Learning

- PCA (1901), kernel PCA
- Multi-dimensional Scaling (1952)
- Maximum Variance Unfolding, Colored MVU
- Mixture of PCA, Mixture of Factor Analyzers
- Local Linear Embedding (2000)
- Isomap (2000), C-Isomap
- Hessian Eigenmaps
- Laplacian Eigenmaps (2003)
- Local Tangent Space Alignment
- stochastic neighbor embedding (2003)
- t-distributed stochastic neighbor embedding (2008)

- … and many more

Principal Component Analysis

Idea:

Given data points in a D-dimensional space,
project them into a lower dimensional space while preserving as much "information" as possible.

Examples:

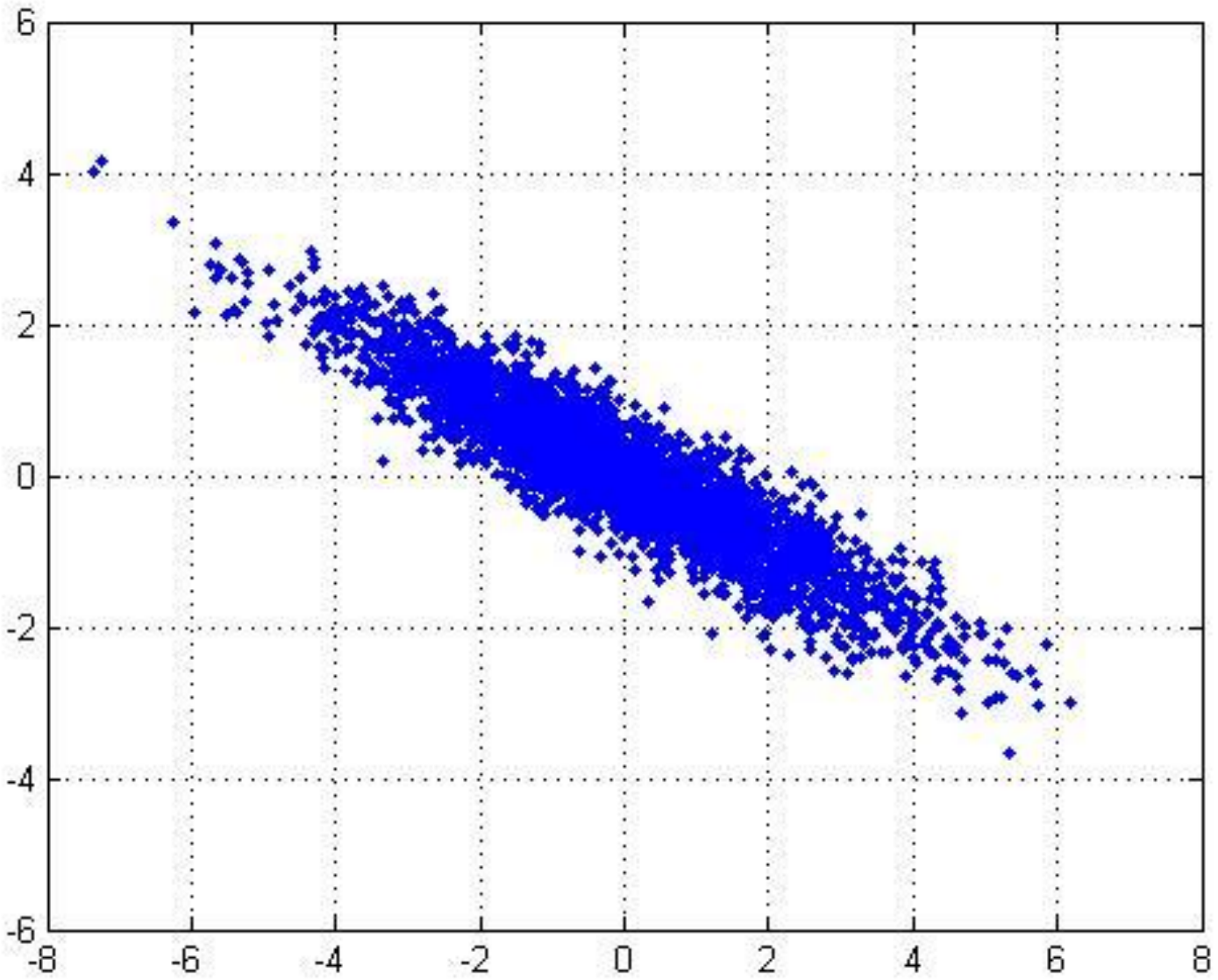
- Find best planar approximation of 3D data
- Find best 12-D approximation of 10^4-D data

Principal Component Analysis

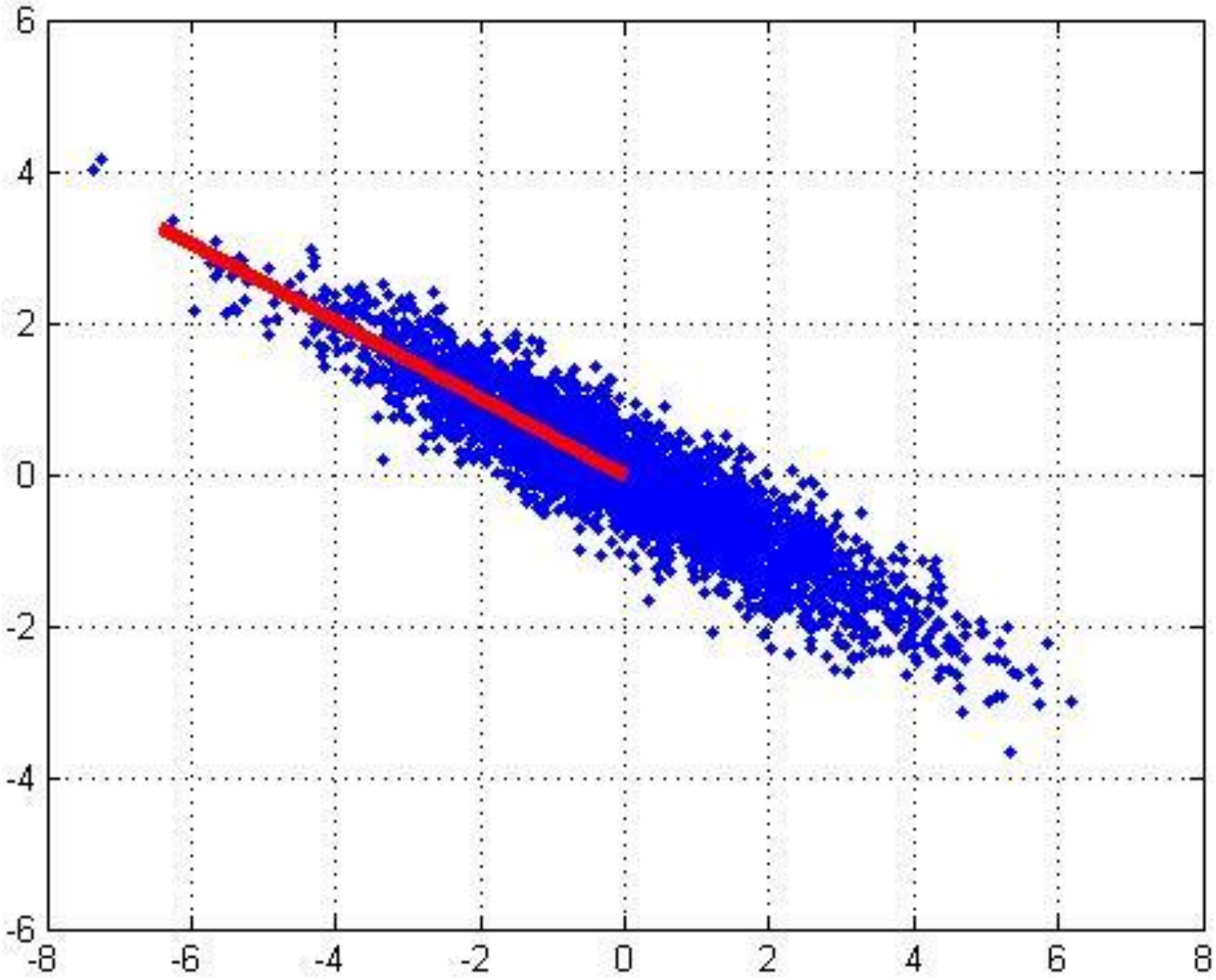
Properties:

- **PCA Vectors** originate from the center of mass.
- Principal component #1: points in the direction of the **largest variance**.
- Each subsequent principal component
  - is **orthogonal** to the previous ones, and
  - points in the directions of the **largest variance of the residual subspace**

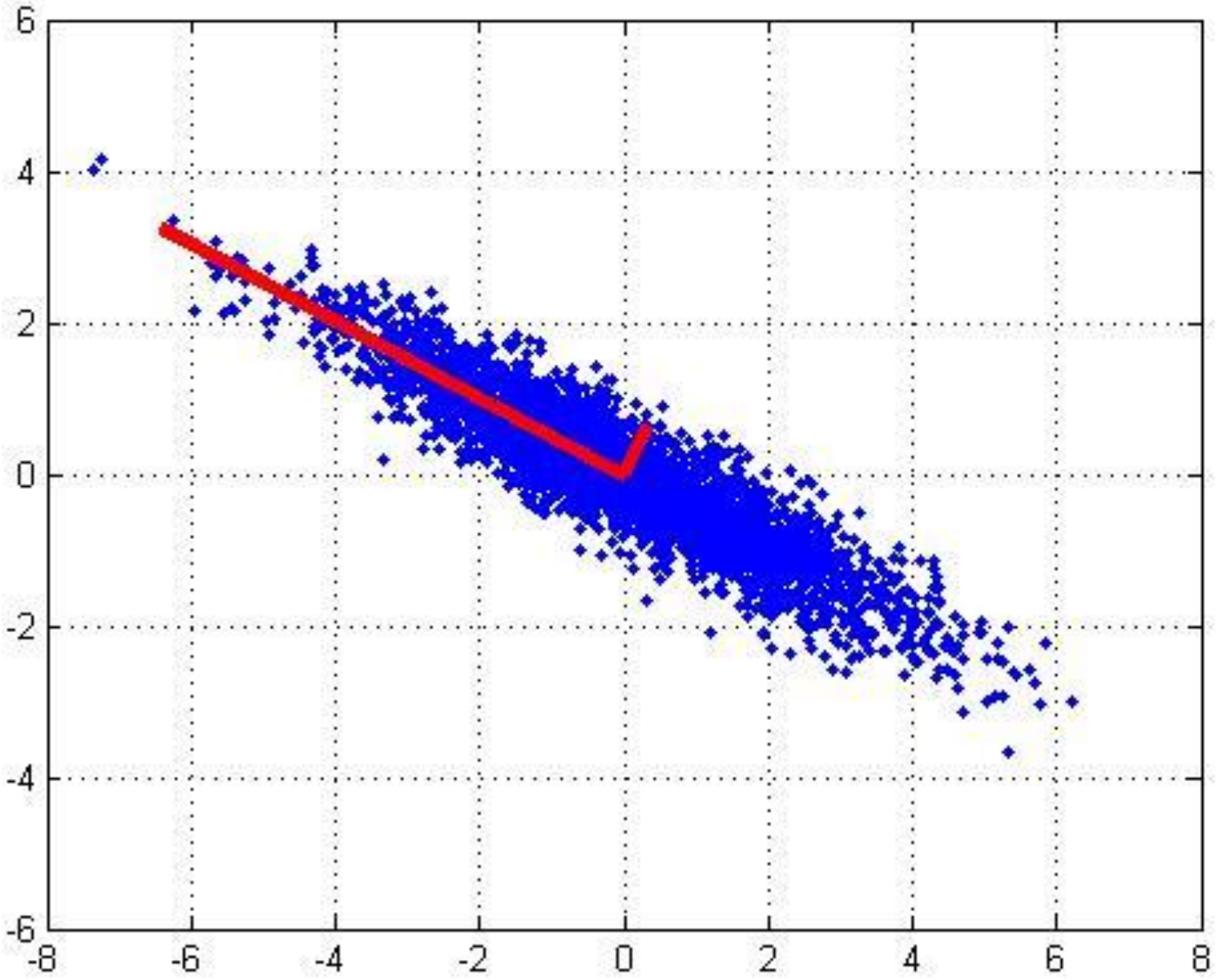
2D Gaussian dataset



1st PCA axis



2nd PCA axis



PCA algorithm I (sample covariance matrix)

- Given data $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, compute covariance matrix Σ

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$$

where

$$\bar{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i$$

- basis vectors = the eigenvectors of Σ

- Larger eigenvalue $\Rightarrow$ more important eigenvectors

PCA algorithm I (sample covariance matrix)

PCA algorithm($\mathbf{X}$, k): top k eigenvalues/eigenvectors

% $\underline{\mathbf{X}}$ = D × m data matrix,
% … each data point $\mathbf{x}_i$ = column vector, i=1..m

- $\underline{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i$
- $\mathbf{X} \leftarrow$ subtract mean $\underline{\mathbf{x}}$ from each column vector $\mathbf{x}_i$ in $\underline{\mathbf{X}}$
- $\Sigma \leftarrow \mathbf{X}\mathbf{X}^T$ … covariance matrix of $\mathbf{X}$
- $\{ \lambda_i, \mathbf{u}_i \}_{i=1\ldots D}$ = eigenvectors/eigenvalues of Σ
  ... $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_D$
- Return $\{ \lambda_i, \mathbf{u}_i \}_{i=1..k}$
  % top k PCA components

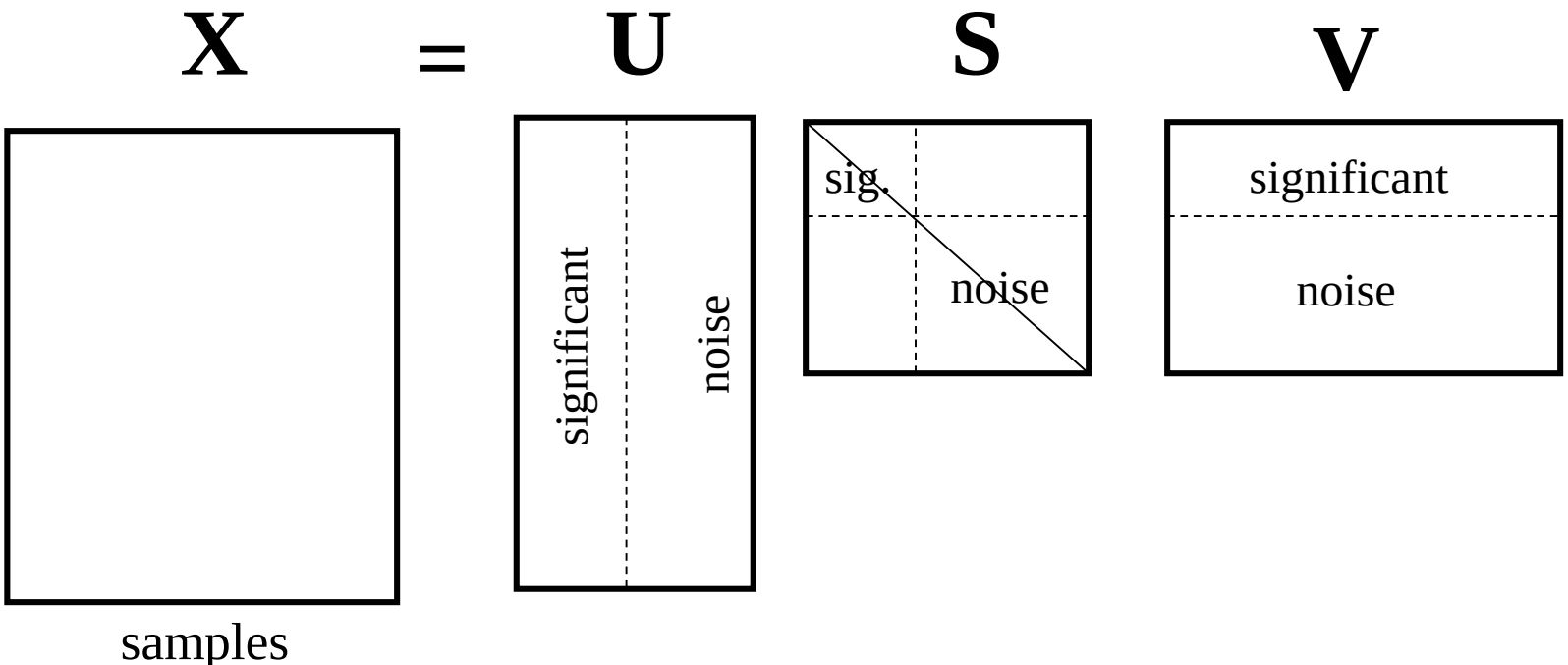
PCA algorithm II (SVD of the data matrix)

Singular Value Decomposition of the **centered** data matrix **X**.

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m] \in \mathbb{R}^{D \times m},$$

m: number of instances,
D: dimension

$$\mathbf{X}_{\text{features} \times \text{samples}} = \mathbf{USV}$$



PCA algorithm II

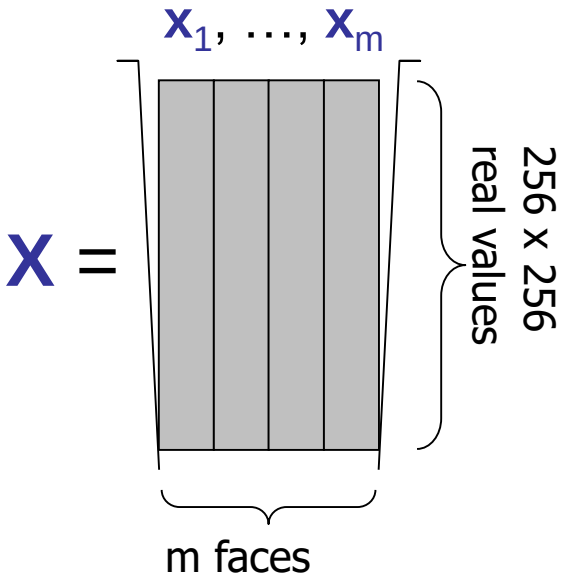
- **Columns of U**
  - the principal vectors, $\{ \mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(k)} \}$
  - orthogonal and has unit norm – so $U^T U = I$
  - Can reconstruct the data using linear combinations of $\{ \mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(k)} \}$

- **Matrix S**
  - Diagonal
  - Shows importance of each eigenvector

- **Matrix V**
  - The coefficients for reconstructing the samples

Applying PCA: Eigenfaces

- Example data set: Images of faces
  - Eigenface approach
    [Turk & Pentland], [Sirovich & Kirby]
- Each face **x** is …
  - 256 × 256 values (luminance at location)
  - **x** in $\Re^{256\times256}$ (view as 64K dim vector)
- Form **X** = [$\mathbf{x}_1$, …, $\mathbf{x}_m$] **centered** data mtx
- Compute $\Sigma = \mathbf{X}\mathbf{X}^T$
- Problem: Σ is 64K × 64K … HUGE!!!



Computational Complexity

- Suppose m instances, each of size D
  - Eigenfaces: $m=500$ faces, each of size $D=64K$
- Given $D \times D$ covariance matrix Σ, can compute
  - all D eigenvectors/eigenvalues in $O(D^3)$
  - first k eigenvectors/eigenvalues in $O(k\ D^2)$

- If $D=64K$, this is EXPENSIVE!

A Clever Workaround

- Note that $m << 64K$
- Use $\mathbf{L = X^T X}$ instead of $\mathbf{\Sigma = X X^T}$
- If $\mathbf{v}$ is eigenvector of $\mathbf{L}$,
  then $\mathbf{Xv}$ is eigenvector of $\mathbf{\Sigma}$

Proof:

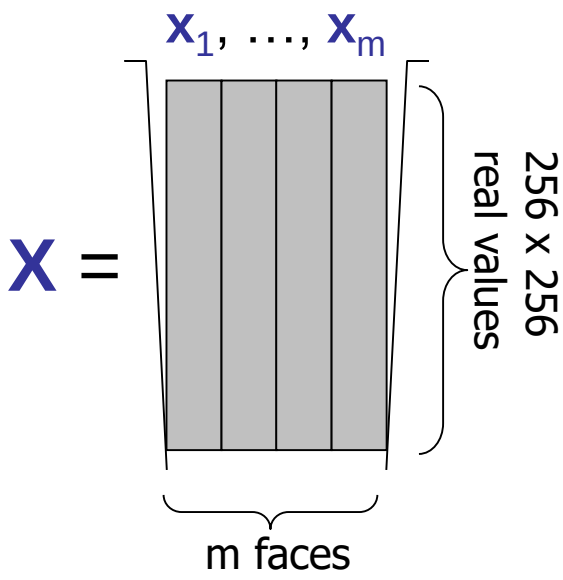
$$\mathbf{L\ v} = \gamma\ \mathbf{v}$$

$$\mathbf{X^T X\ v} = \gamma\ \mathbf{v}$$

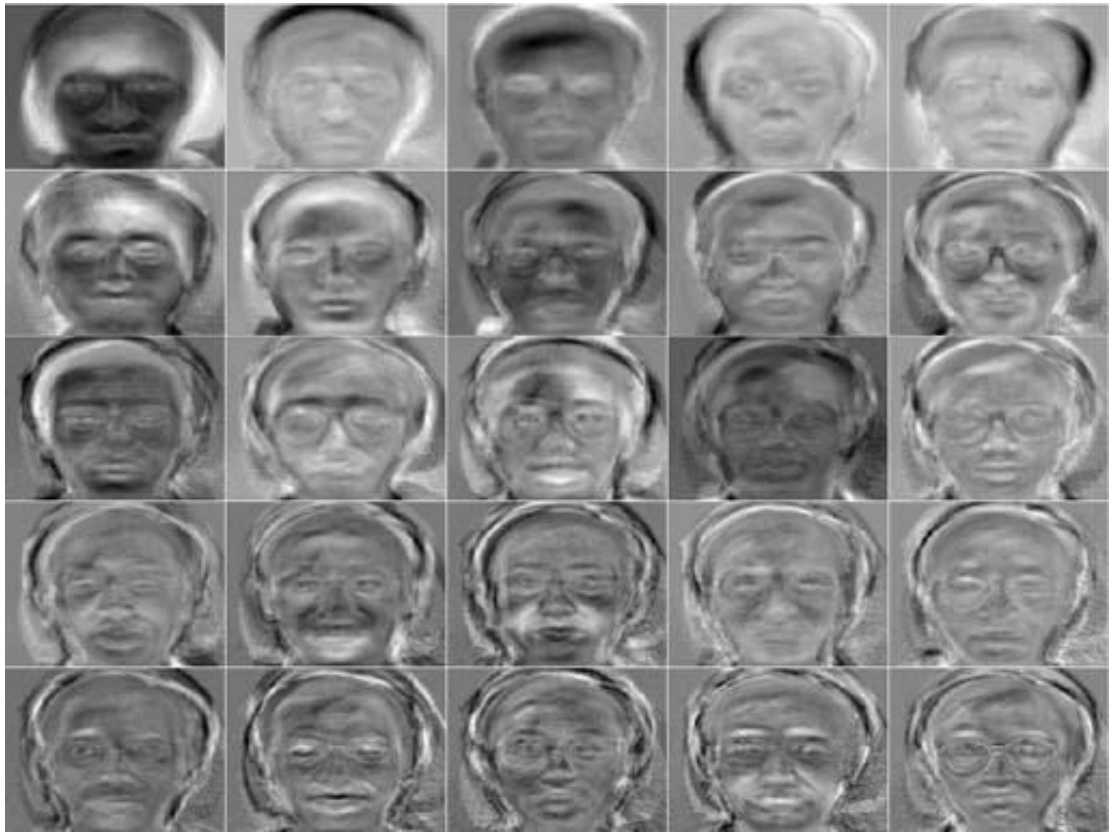
$$\mathbf{X\ (X^T X\ v)} = \mathbf{X}(\gamma\ \mathbf{v}) = \gamma\ \mathbf{Xv}$$

$$\mathbf{(X X^T) X\ v} = \gamma\ (\mathbf{Xv})$$

$$\mathbf{\Sigma\ (Xv)} = \gamma\ (\mathbf{Xv})$$

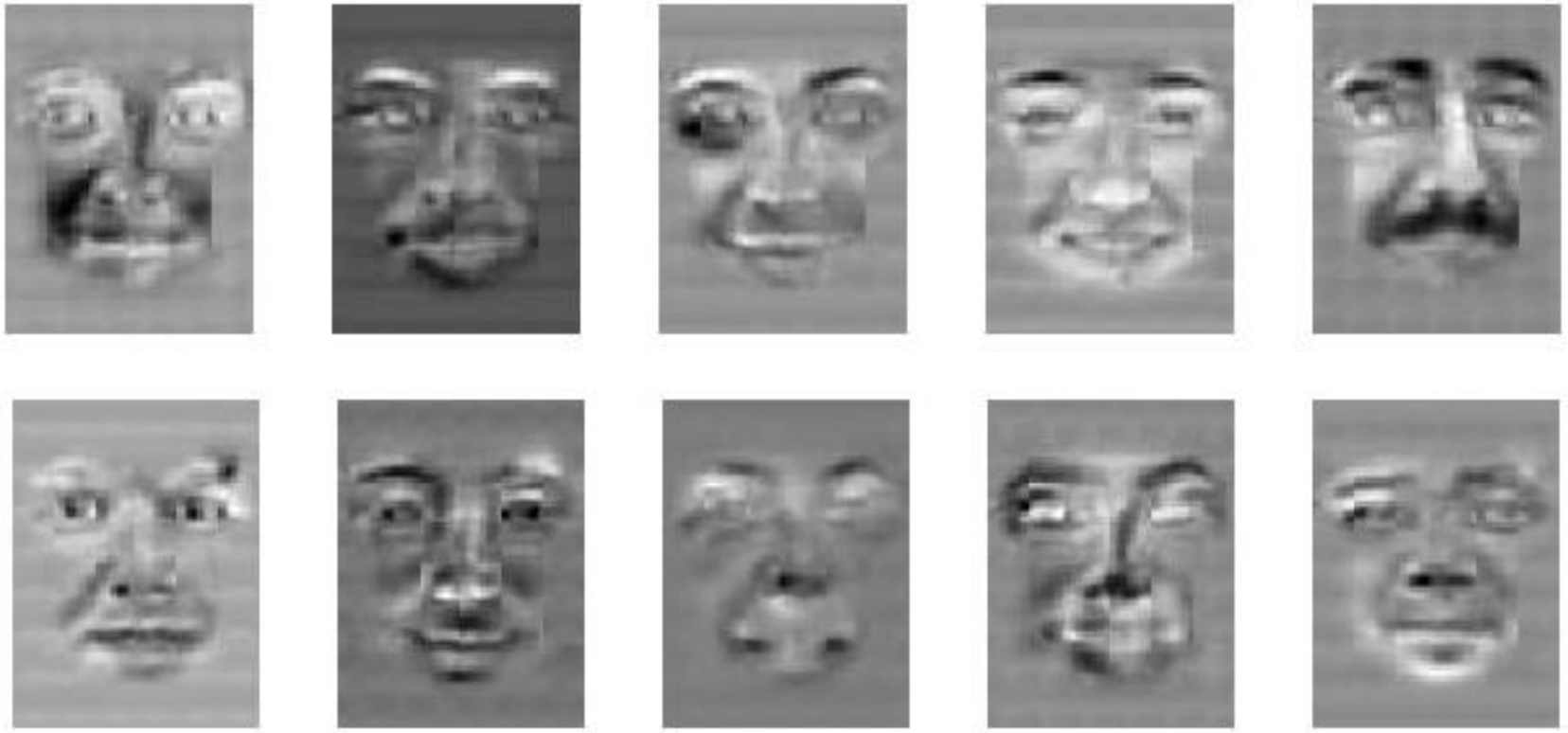


Principle Components



Principal Component Analysis (PCA)

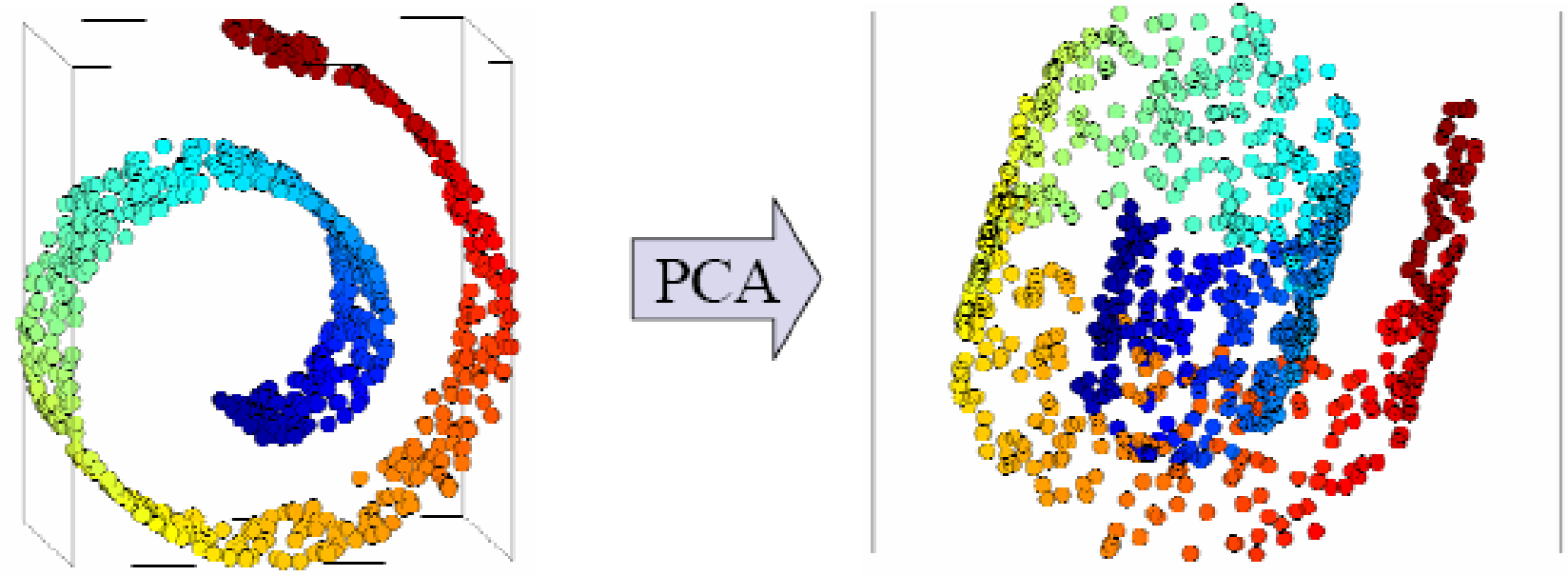
Eigenfaces



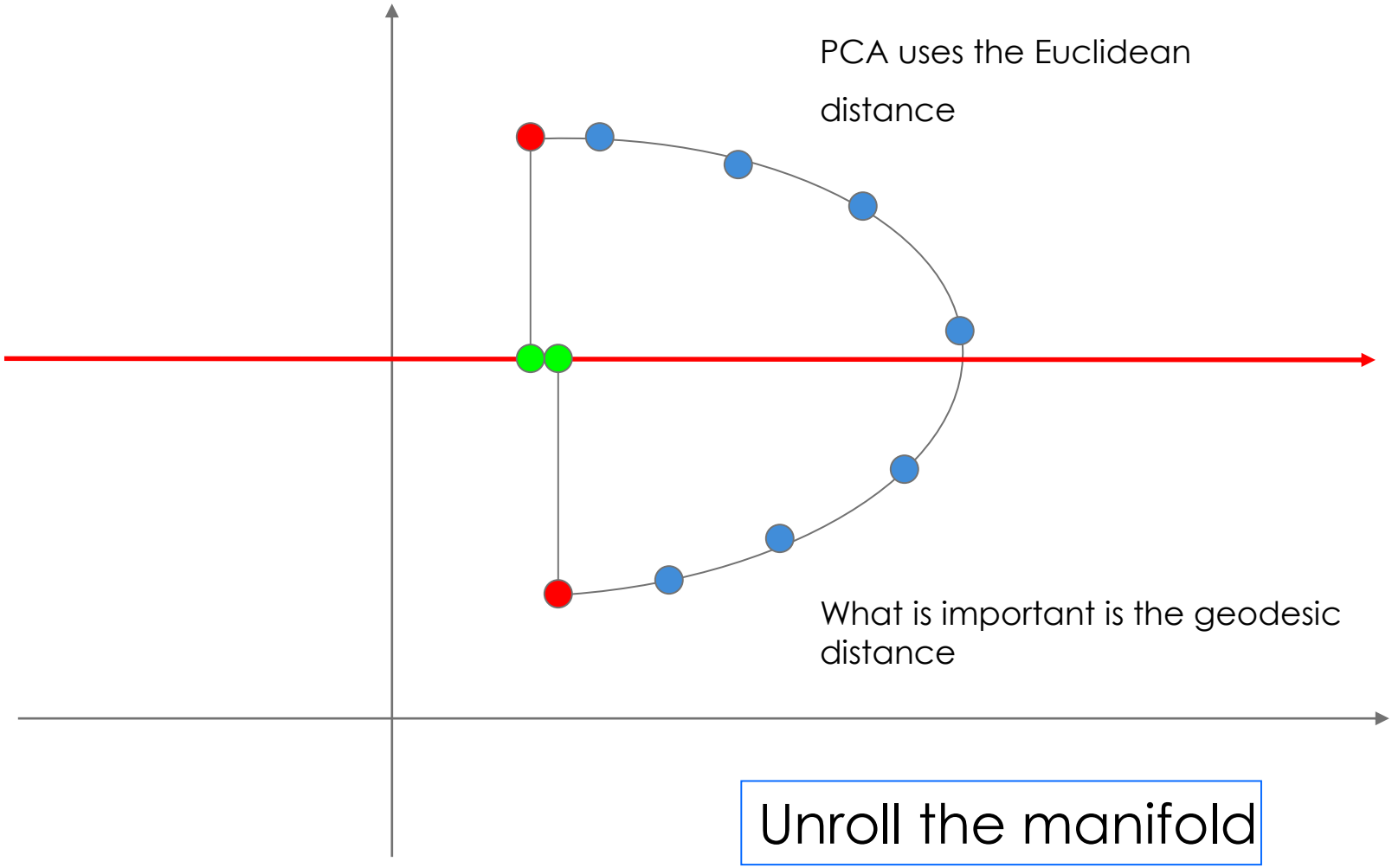
Issues with PCA

PCA is a linear method:

it fails to find the nonlinear structure in the data



Issues with PCA



Unroll the manifold

Local Linear Embedding

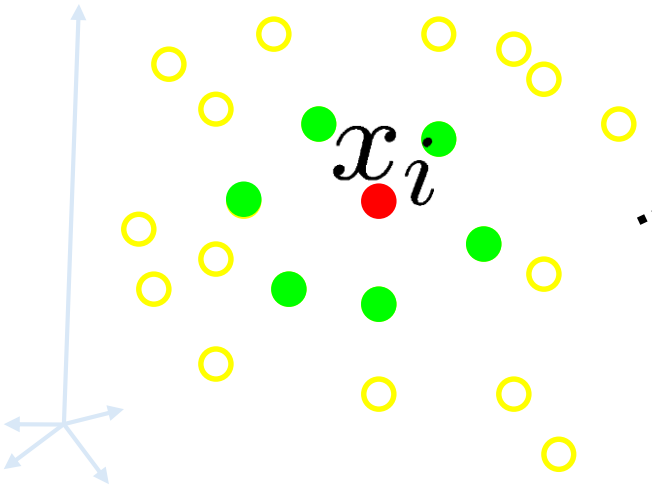
Nonlinear dimensionality reduction by locally linear embedding.
Sam Roweis & Lawrence Saul.
Science, v.290 no 5500, Dec.22, 2000. pp.2323--2326.

Local Linear Embedding

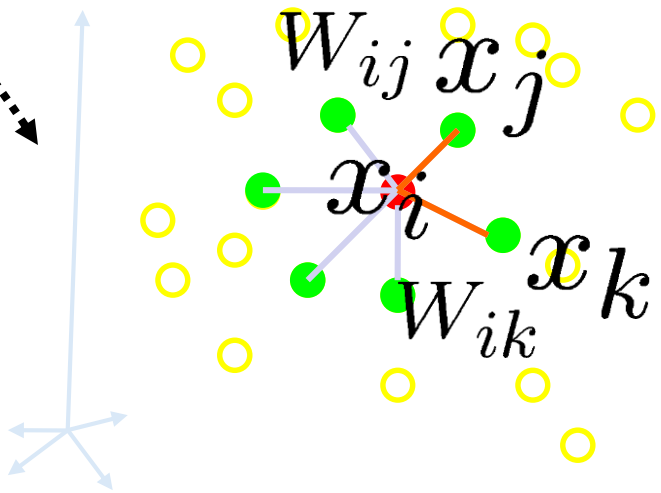
Assumption: manifold is approximately "linear" when viewed locally. Data:

$$X = [x_1, \ldots, x_m] \in \mathbb{R}^{D \times m}$$

$$W = \arg\min_W \sum_{i=1}^{m} \|x_i - \sum_{j=1}^{m} W_{ij} x_j\|^2$$



1. select neighbors (epsilon or kNN)



2. reconstruct x_i with weighted linear sum

Local Linear Embedding

Step 1.

$$W = \arg\min_{W} \sum_{i=1}^{m} \|x_i - \sum_{j=1}^{m} W_{ij} x_j\|^2$$

Subject to $\sum_j W_{ij} = 1$, $\forall i$,
and $W_{ij} = 0$ if x_j is not neighbor of x_i.

Local Linear Embedding

Step 2. Given the weights W, find the embedded points:

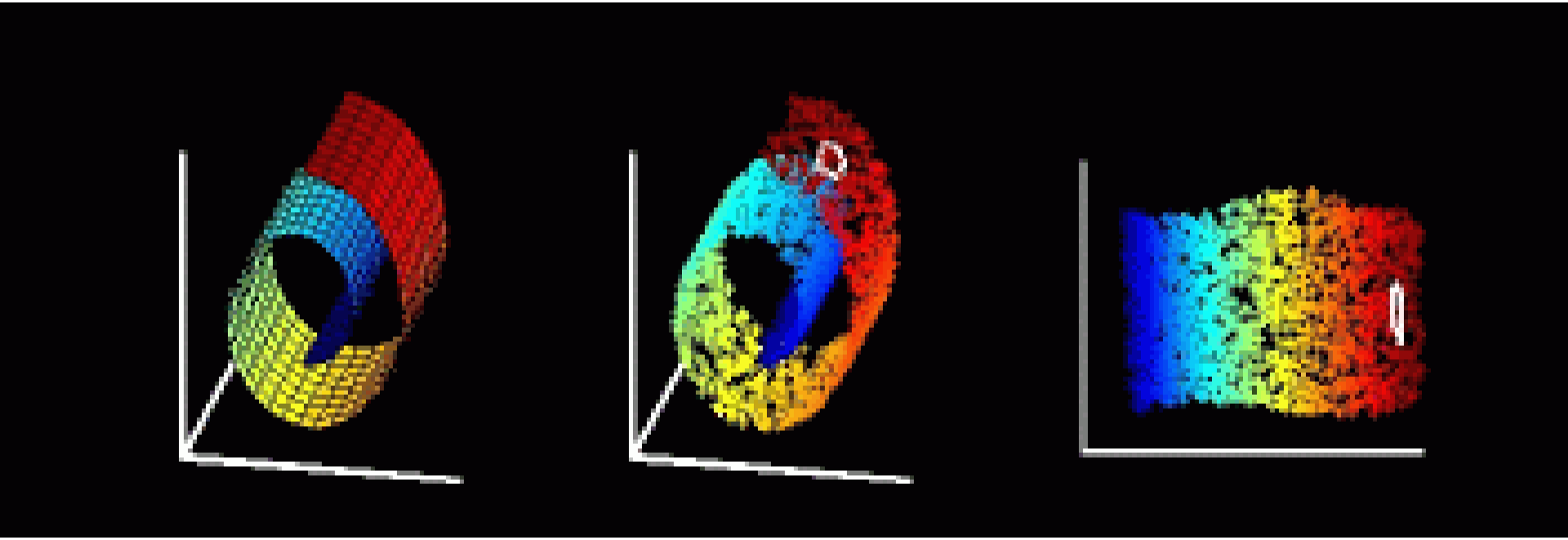
$$[z_1, \ldots, z_m] = \arg \min_{[z_1,\ldots,z_m]} \sum_{i=1}^{m} \|z_i - \sum_{j=1}^{m} W_{ij} z_j\|^2$$

Subject to $\sum_i z_i = 0$
and unit covariance matrix.

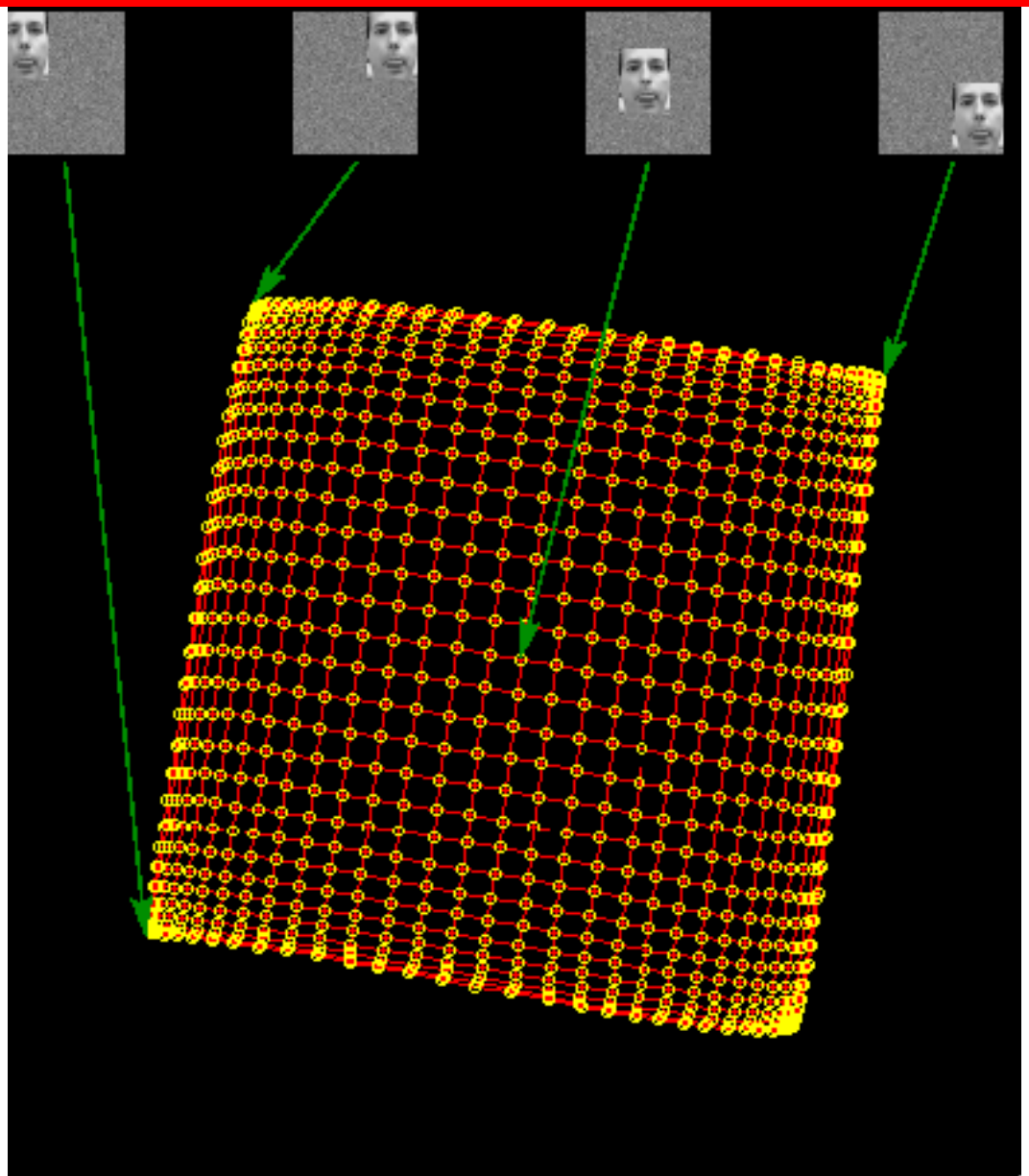
The same weights that reconstruct the data points in D dimensions should also reconstruct the points in d dimensions.

The weights characterize the intrinsic geometric properties of each neighborhood.

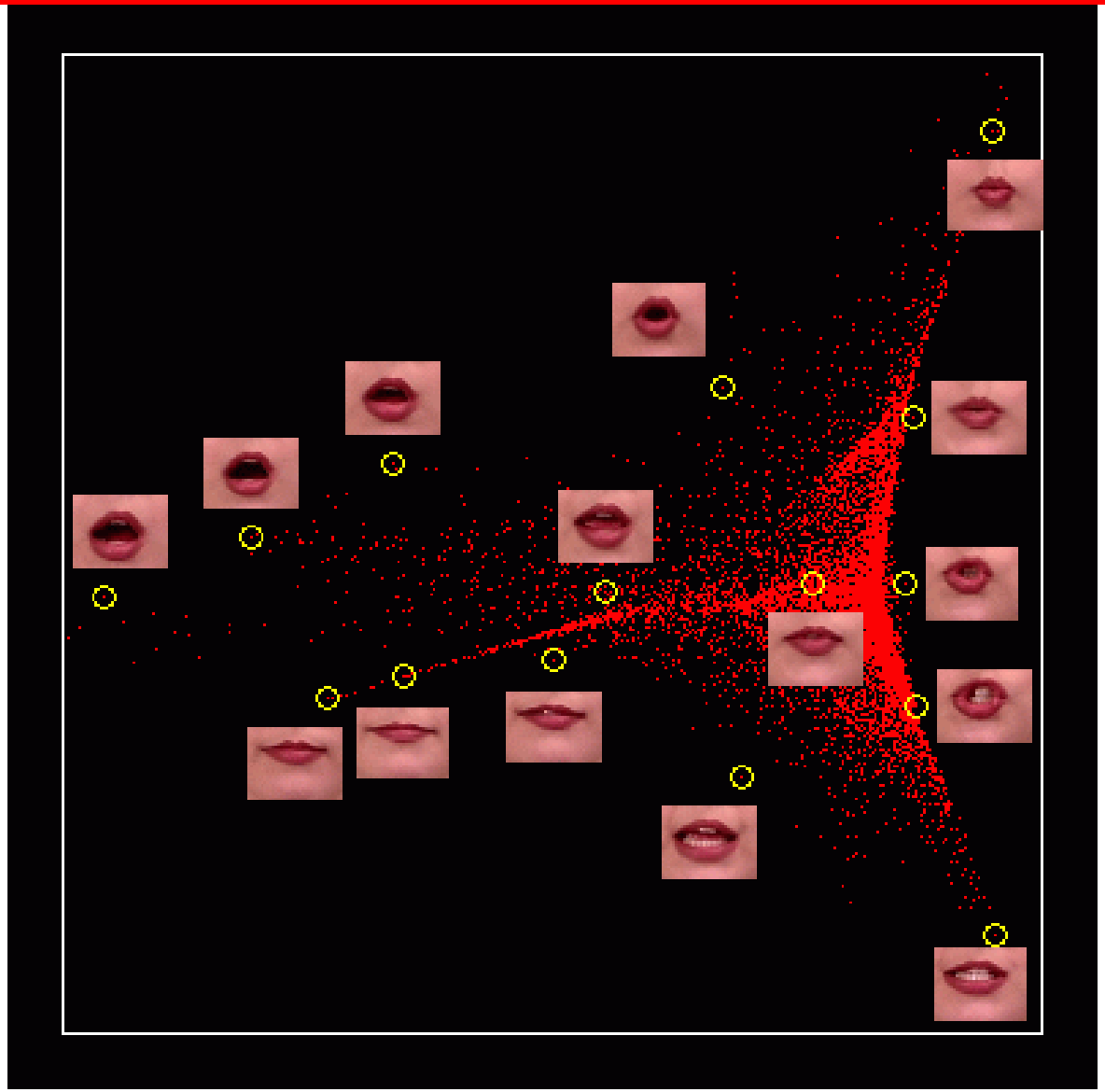
Applications



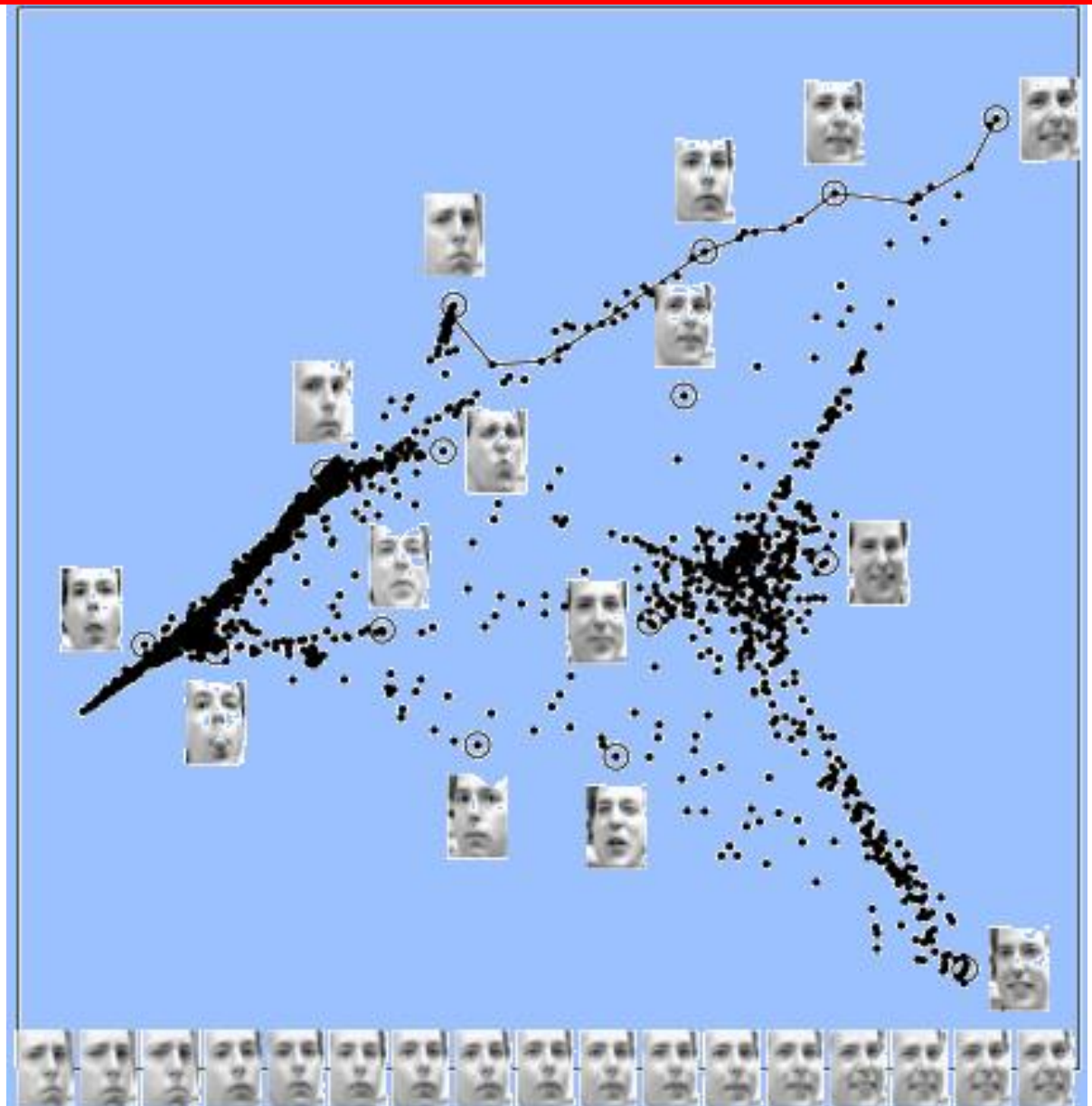
Applications



Applications



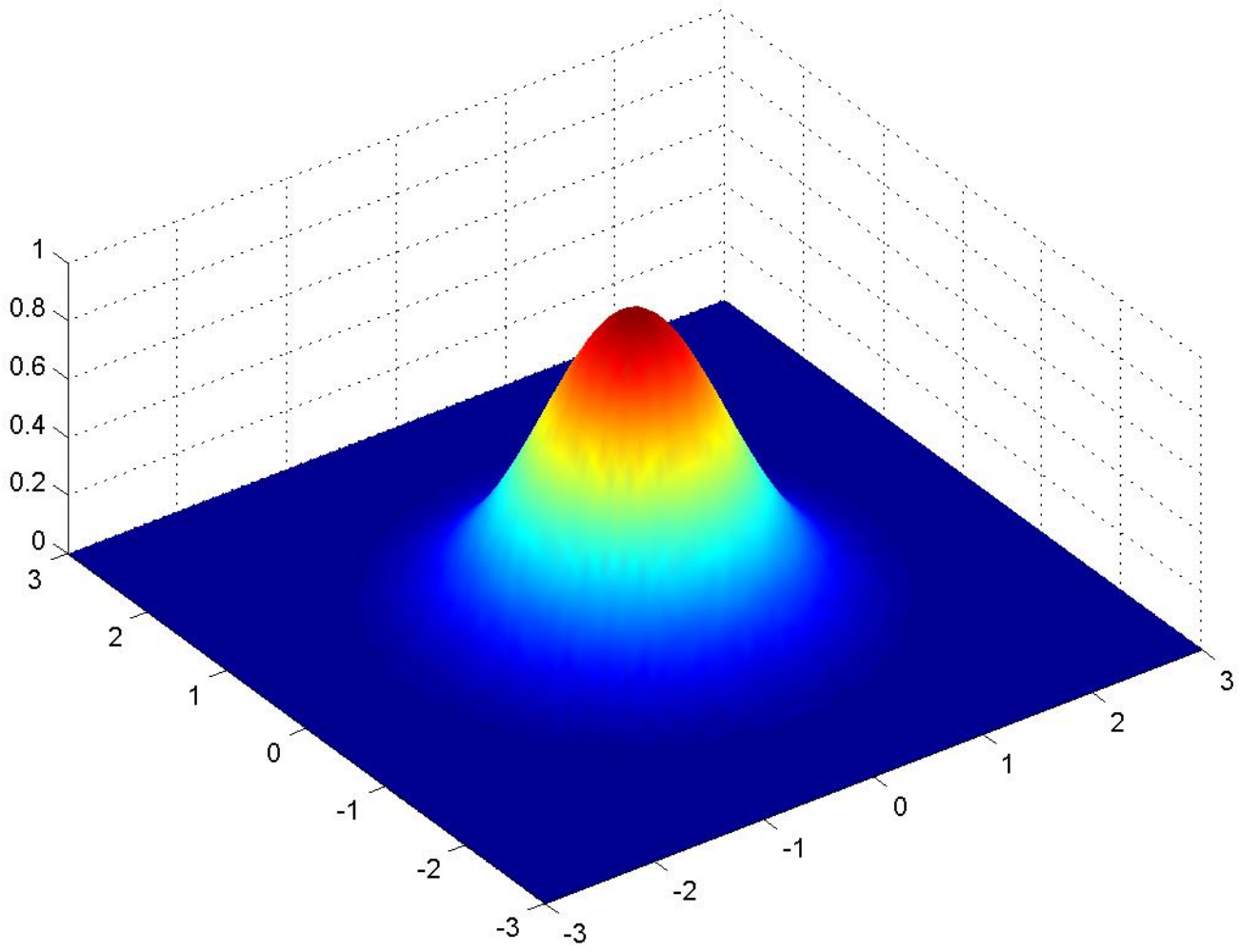
Applications



t-distributed stochastic neighbor embedding

L.J.P. van der Maaten and G.E. Hinton. **Visualizing High-Dimensional Data Using t-SNE**. *Journal of Machine Learning Research* 9(Nov):2579-2605, 2008

Gaussian Distribution



$$p(x) \sim \exp(-\|\mathbf{x} - \mu\|^2/2\sigma^2)$$

t-distributed stochastic neighbor embedding

Step 1 Given a set of N high-dimensional objects $\mathbf{x}_1, \ldots, \mathbf{x}_N$, t-SNE computes probabilities p_{ij} that are proportional to the distance of objects $\mathbf{x}_i$ and $\mathbf{x}_j$

$$\text{Let } p_{j|i} \doteq \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}$$

Informal meaning:

$p_{j|i}$ = Probabiltiy that x_i would pick x_j as its neighbor if neighbors were picked in proportion to Gaussian density centered at x_i.

Observation: $\sum_{\substack{j=1 \\ j\neq i}}^{N} p_{j|i} = 1$

t-distributed stochastic neighbor embedding

Let $p_{ij} \doteq \frac{p_{j|i} + p_{i|j}}{2N}$ Make the distirbution symmetric in i and j.

Observation:

$$\sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} p_{j|i} = N$$

$$\sum_{j=1}^{N} \sum_{\substack{i=1 \\ i \neq j}}^{N} p_{i|j} = N$$

$$\sum_{\substack{i,j=1 \\ j \neq i}}^{N} p_{ij} = \frac{2N}{2N} = 1$$

$\Rightarrow$ This is a valid 2 dimensional joint discrete distribution on the set $\{(i,j)|i \neq j, 1 \leq, i, j \leq N\}$.

t-distributed stochastic neighbor embedding

Step 2 t-SNE aims to learn d-dimensional map $\mathbf{y}_1, \ldots, \mathbf{y}_N$ (with $\mathbf{y}_i \in \mathbb{R}^d$) that reflects the similarities p_{ij} as well as possible.

It measures similarities q_{ij} between two points in the map $\mathbf{y}_i$ and $\mathbf{y}_j$, using a very similar approach:

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq i}(1 + \|\mathbf{y}_i - \mathbf{y}_k\|^2)^{-1}}$$

Observation:

$$\sum_{\substack{i,j=1 \\ j \neq i}}^{N} q_{ij} = 1$$

$\Rightarrow$ This is a valid 2 dimensional joint discrete distribution on the set $\{(i,j)|i \neq j, 1 \leq, i, j \leq N\}$.

t-distributed stochastic neighbor embedding

Step 3 Find the best $\mathbf{y}_1, \ldots, \mathbf{y}_N$ by minimizing the KL divergence:

$$\min_{\mathbf{y}_1,\cdots,\mathbf{y}_N} KL(P||Q) = \min_{\mathbf{y}_1,\cdots,\mathbf{y}_N} \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

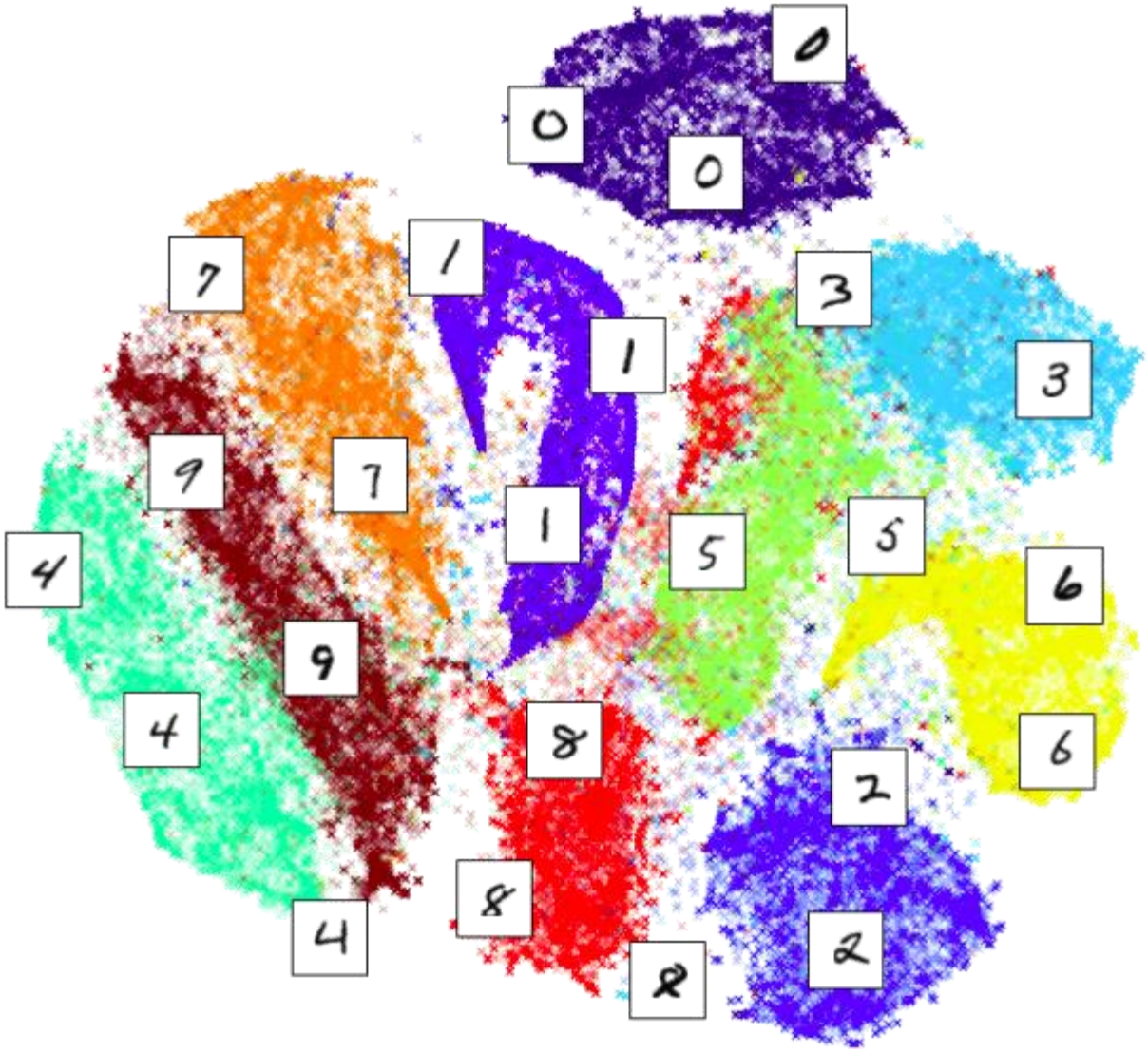
t-distributed stochastic neighbor embedding

Algorithm Summary

1. t-SNE constructs a probability distribution over pairs of high-dimensional objects such that:
   - similar objects have a high probability of being picked,
   - dissimilar points have an extremely small probability of being picked.

2. t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback–Leibler divergence between the two distributions with respect to the locations of the points in the map.

t-SNE on MNIST



Code Examples

- Audio tSNE
  https://github.com/bapoczos/ArtML/blob/master/Embeddings/audio-tsne.ipynb
- Image Path
  https://github.com/bapoczos/ArtML/blob/master/Embeddings/image-path.ipynb
- Image Search
  https://github.com/bapoczos/ArtML/blob/master/Embeddings/image-search.ipynb
- Image tSNE
  https://github.com/bapoczos/ArtML/blob/master/Embeddings/image-tsne.ipynb
- Word2Vec tSNE
  https://github.com/bapoczos/ArtML/blob/master/Embeddings/word2vec_tsne.ipynb

Multi-dimensional Scaling

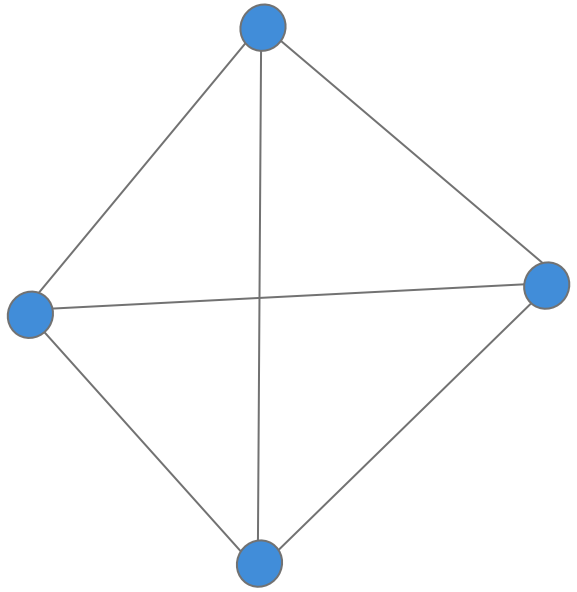
Multi-dimensional Scaling

- In PCA we are given a set of points

$$X = [x_1, \ldots, x_N] \in \mathbb{R}^{D \times N}$$

- In MDS we are given pairwise distances instead of the actual data points.

- Question: If we only know the pairwise distances, can we also preserve the geometric structure (i.e. the inner products between points)?



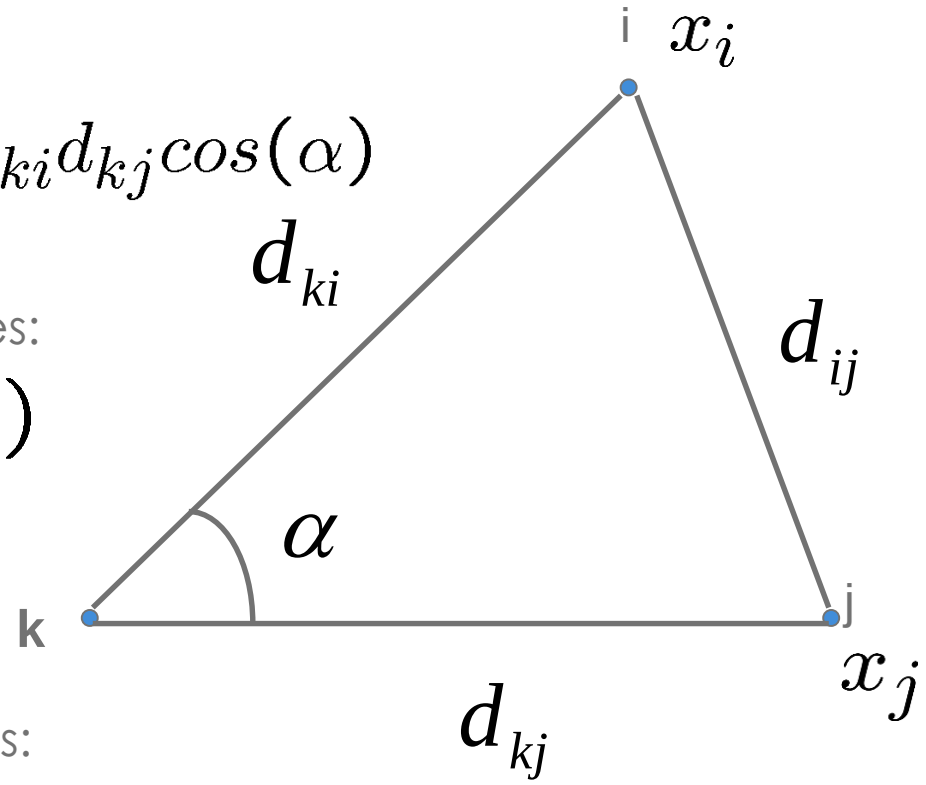
From Distances to Inner Products

Cosine law (angle from distances):

$$d_{ij}^2 = d_{ki}^2 + d_{kj}^2 - 2d_{ki}d_{kj}cos(\alpha)$$

Inner product from angle and distances:

$$b_{ij} = d_{ki}d_{kj}cos(\alpha)$$



Therefore, inner product from distances:

$$b_{ij} \doteq \langle x_i - x_k, x_j - x_k \rangle = \frac{1}{2}(d_{ki}^2 + d_{kj}^2 - d_{ij}^2)$$

From Distances to Inner Products

Similarly:

Center the data and then calculate $\langle x_i, x_j \rangle$

$$\langle x_i, x_j \rangle = G_{ij} = -\frac{1}{2}\left[d_{ij}^2 - \frac{1}{N}\sum_{l=1}^{N} d_{il}^2 - \frac{1}{n}\sum_{m=1}^{n} d_{mj}^2 + \frac{1}{N^2}\sum_{o=1}^{N}\sum_{p=1}^{N} d_{op}^2\right]$$

MDS cost function:

$$J_{MDS}(y_1, \ldots, y_n) = \sum_{i,j}(\langle x_i, x_j \rangle - \langle y_i, y_j \rangle)^2$$

From Distances to Inner Products

MDS algorithm:

Step 1: Build a Gram matrix of inner products

$$X = [x_1, \ldots, x_N] \in \mathbb{R}^{D \times N}$$
$$G = \{\langle x_i, x_j \rangle\}_{i,j} = X^T X \in \mathbb{R}^{N \times N}$$

Step 2: Find a k-rank approximation of G, i.e. calculate thetop k eigenvectors of G:

$$[\psi_1, \ldots, \psi_k] \in \mathbb{R}^{N \times k}$$

with the top k eigenvalues: $\Lambda = diag(\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^{k \times k}$

Step 3: $[y_1, \ldots, y_n] = \Lambda^{1/2}[\psi_1, \ldots, \psi_k]^T \in \mathbb{R}^{k \times N}$

		1 BOST	2 NY	3 DC	4 MIAM	5 CHIC	6 SEAT	7 SF	8 LA	9 DENV
1	BOSTON	0	206	429	1504	963	2976	3095	2979	1949
2	NY	206	0	233	1308	802	2815	2934	2786	1771
3	DC	429	233	0	1075	671	2684	2799	2631	1616
4	MIAMI	1504	1308	1075	0	1329	3273	3053	2687	2037
5	CHICAGO	963	802	671	1329	0	2013	2142	2054	996
6	SEATTLE	2976	2815	2684	3273	2013	0	808	1131	1307
7	SF	3095	2934	2799	3053	2142	808	0	379	1235
8	LA	2979	2786	2631	2687	2054	1131	379	0	1059
9	DENVER	1949	1771	1616	2037	996	1307	1235	1059	0

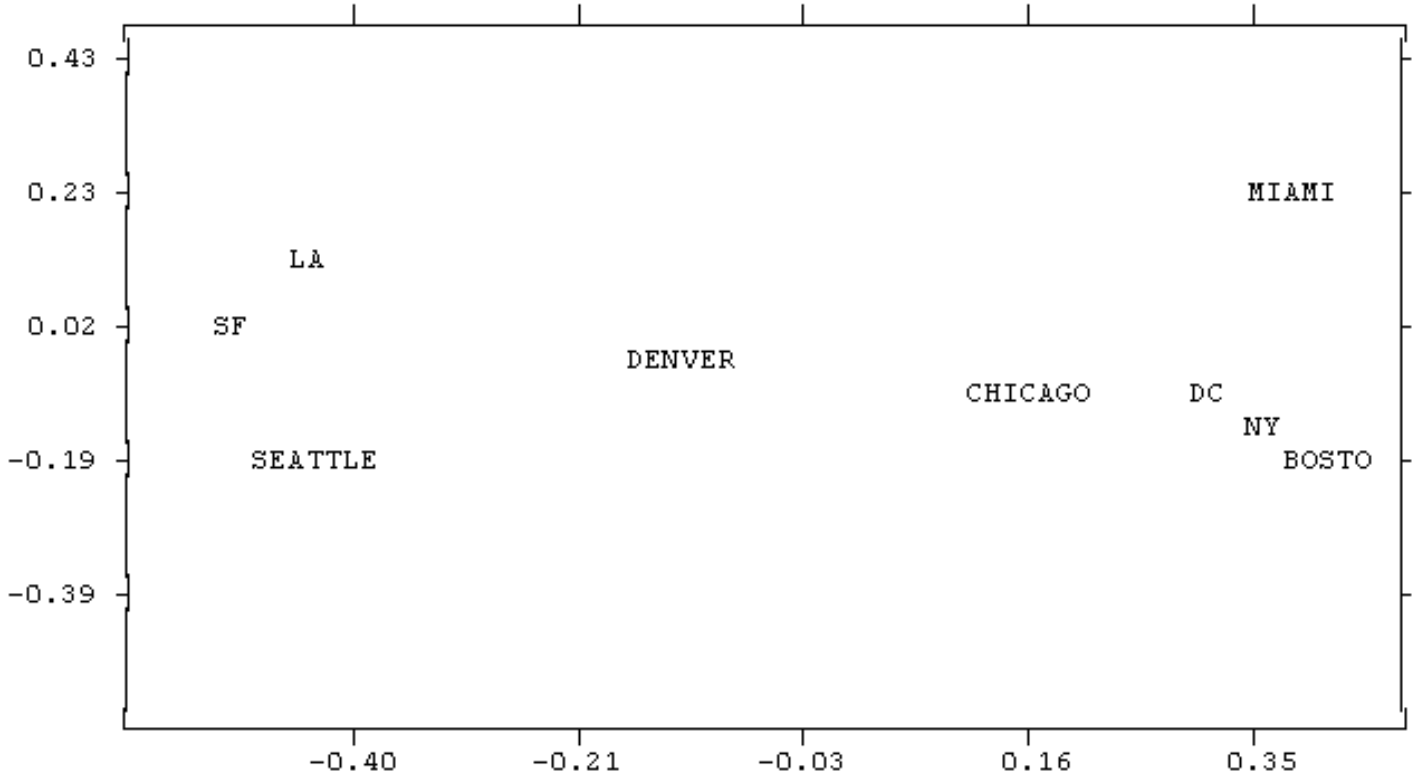


Table 1 Flying Mileages Between 10 American Cities

Atlanta	Chicago	Denver	Houston	Los Angeles	Miami	New York	San Francisco	Seattle	Washington, DC	
0	587	1212	701	1936	604	748	2139	2182	543	Atlanta
587	0	920	940	1745	1188	713	1858	1737	597	Chicago
1212	920	0	879	831	1726	1631	949	1021	1494	Denver
701	940	879	0	1374	968	1420	1645	1891	1220	Houston
1936	1745	831	1374	0	2339	2451	347	959	2300	Los Angeles
604	1188	1726	968	2339	0	1092	2594	2734	923	Miami
748	713	1631	1420	2451	1092	0	2571	2408	205	New York
2139	1858	949	1645	347	2594	2571	0	678	2442	San Francisco
2182	1737	1021	1891	959	2734	2408	678	0	2329	Seattle
543	597	1494	1220	2300	923	205	2442	2329	0	Washington, DC

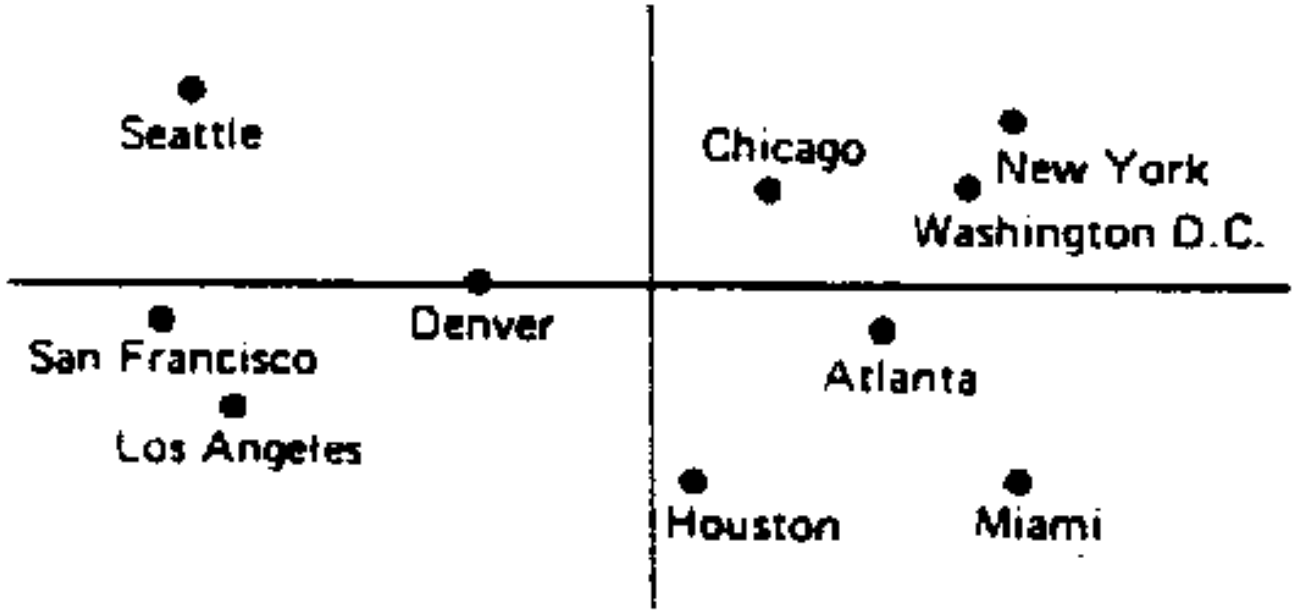


Figure 1 CMDS of flying mileages between 10 American cities.

Isomap

A Global Geometric Framework for Nonlinear Dimensionality Reduction

J. B. Tenenbaum, V. de Silva and J. C. Langford
Science 290 (5500): 2319-2323, 22 December 2000

ISOMAP

Comes from Isometric feature mapping

Step1: Take a data matrix as input.

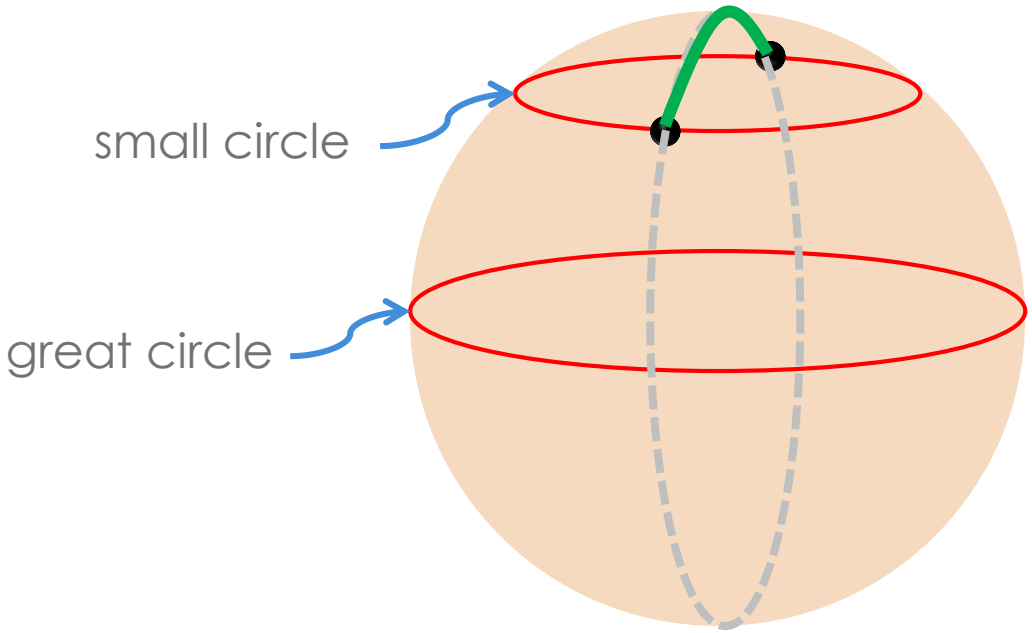
Step2: Estimate geodesic distance between any two points by "a chain of short paths". Use e.g. MST on kNN graphs.

Step3: Perform MDS

Differential Geometry

Geodesic: the shortest curve on a manifold that connects two points on the manifold

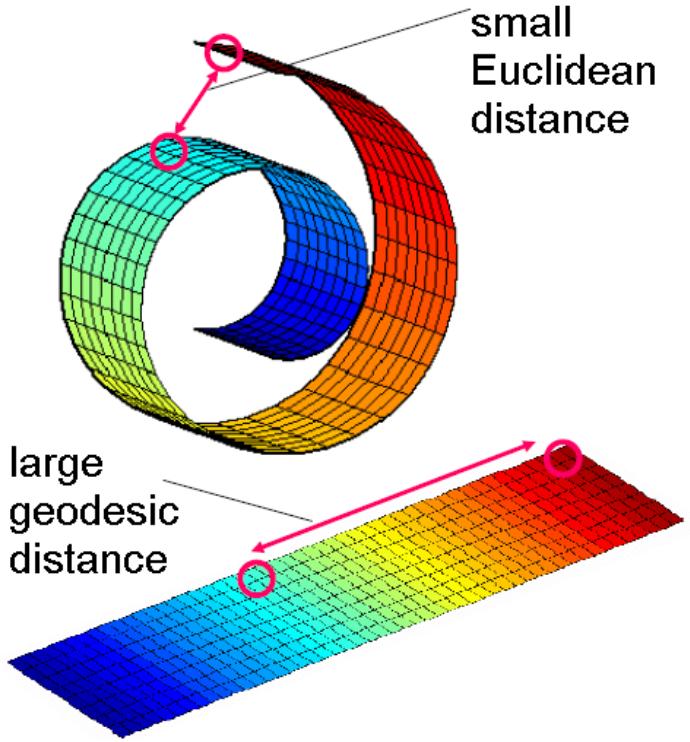
Example (3D sphere)



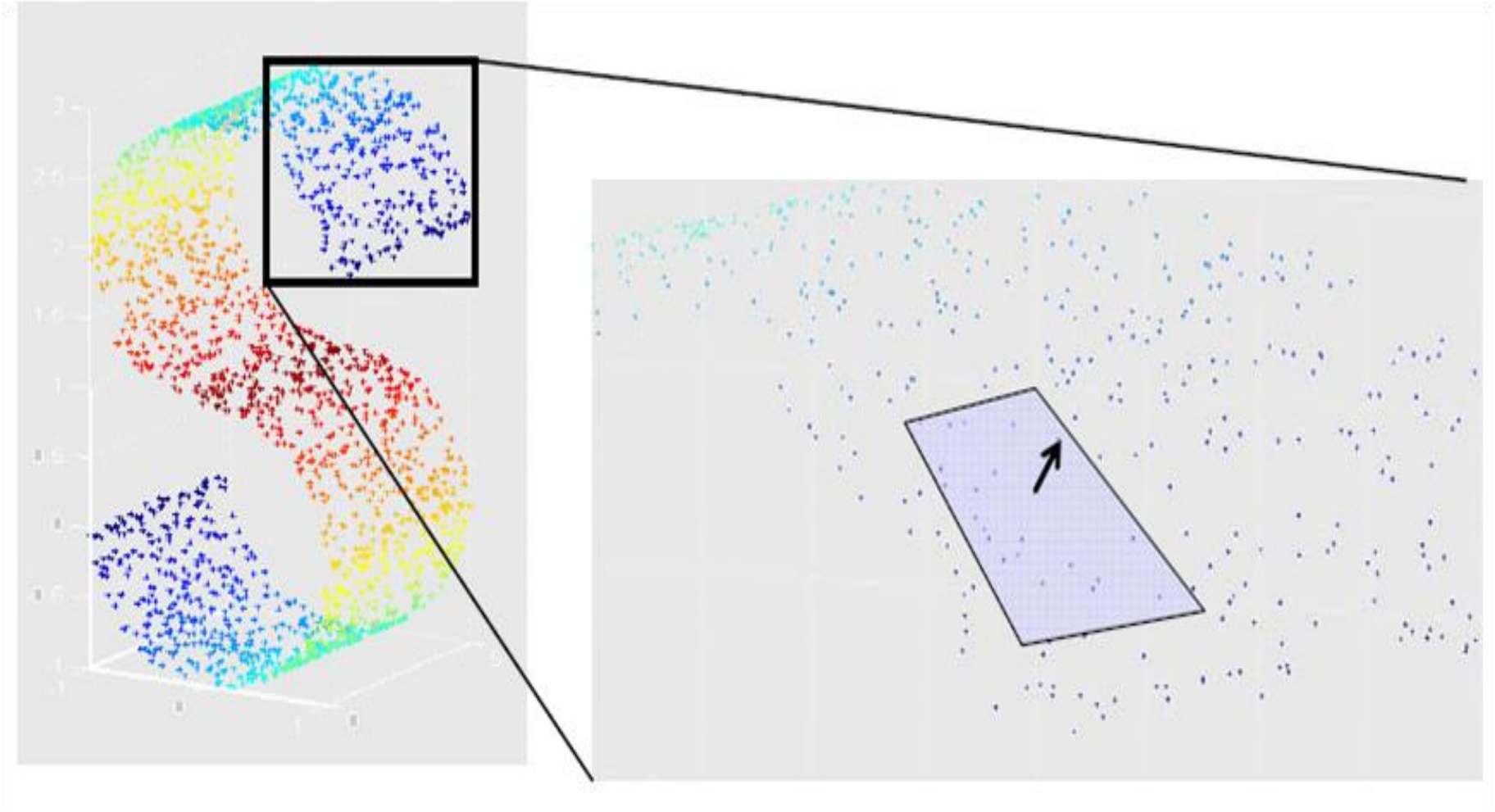
Geodesic distance

Euclidean distance might not be a good measure between two points on a manifold

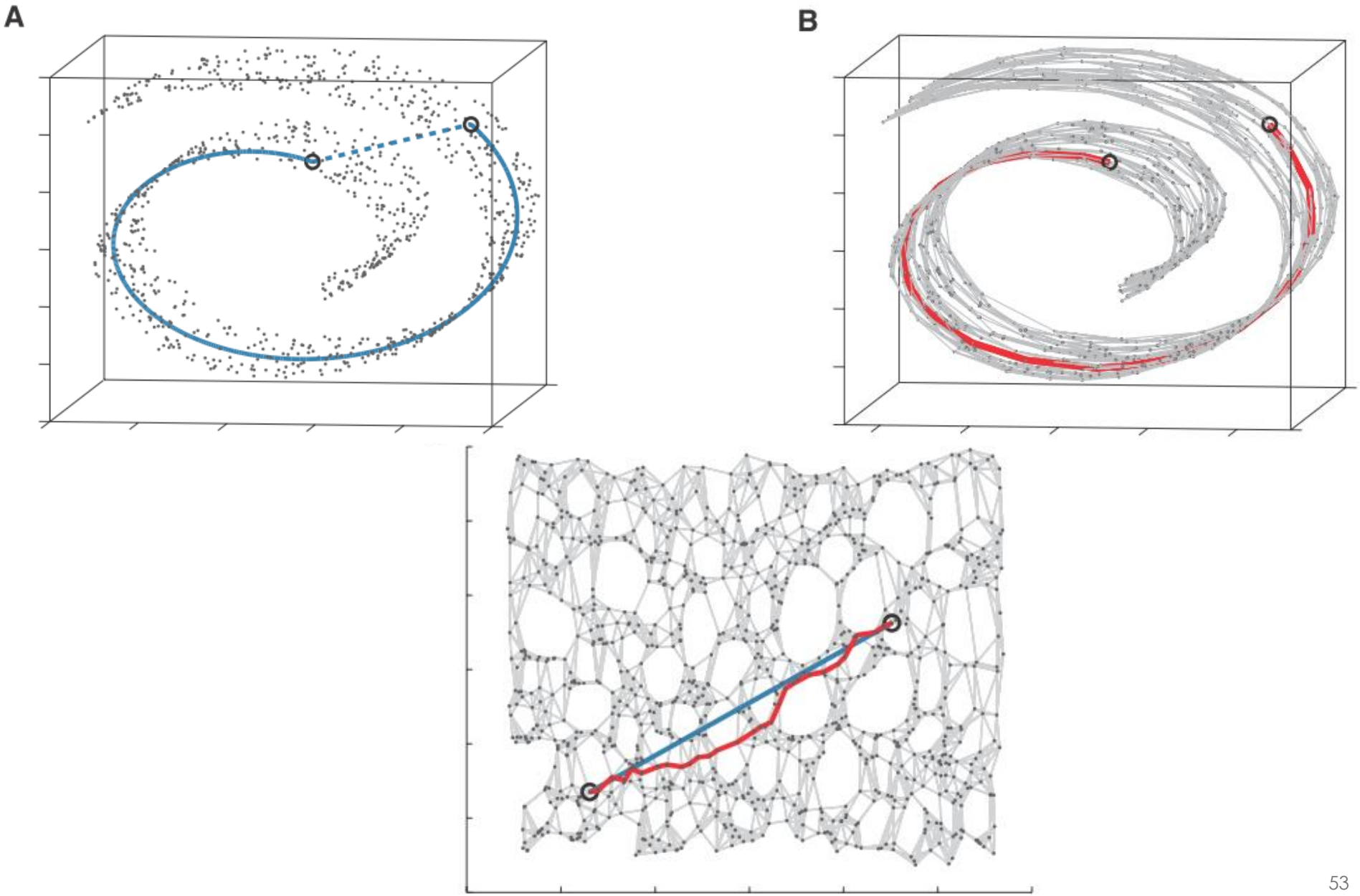
Length of geodesic is more appropriate

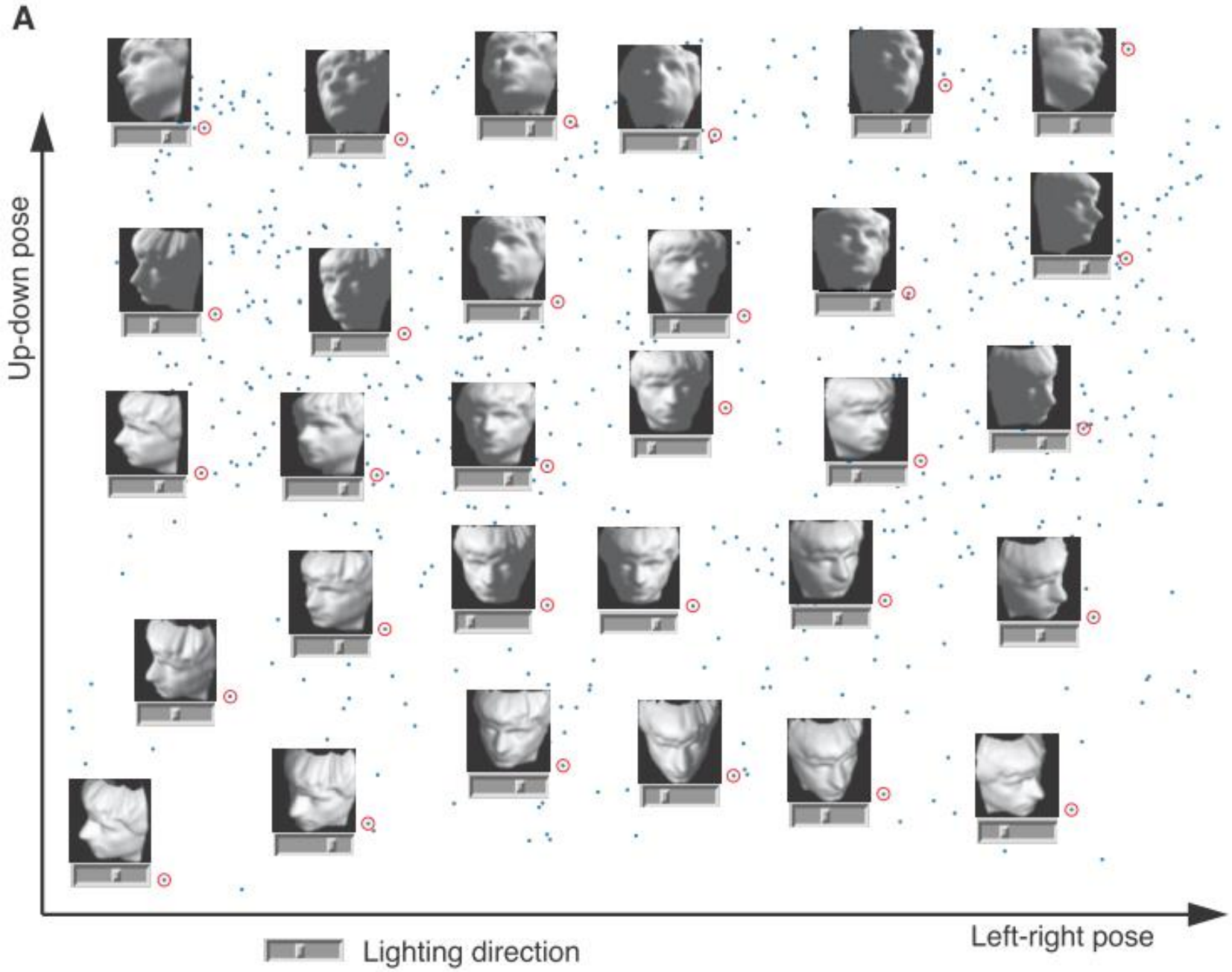


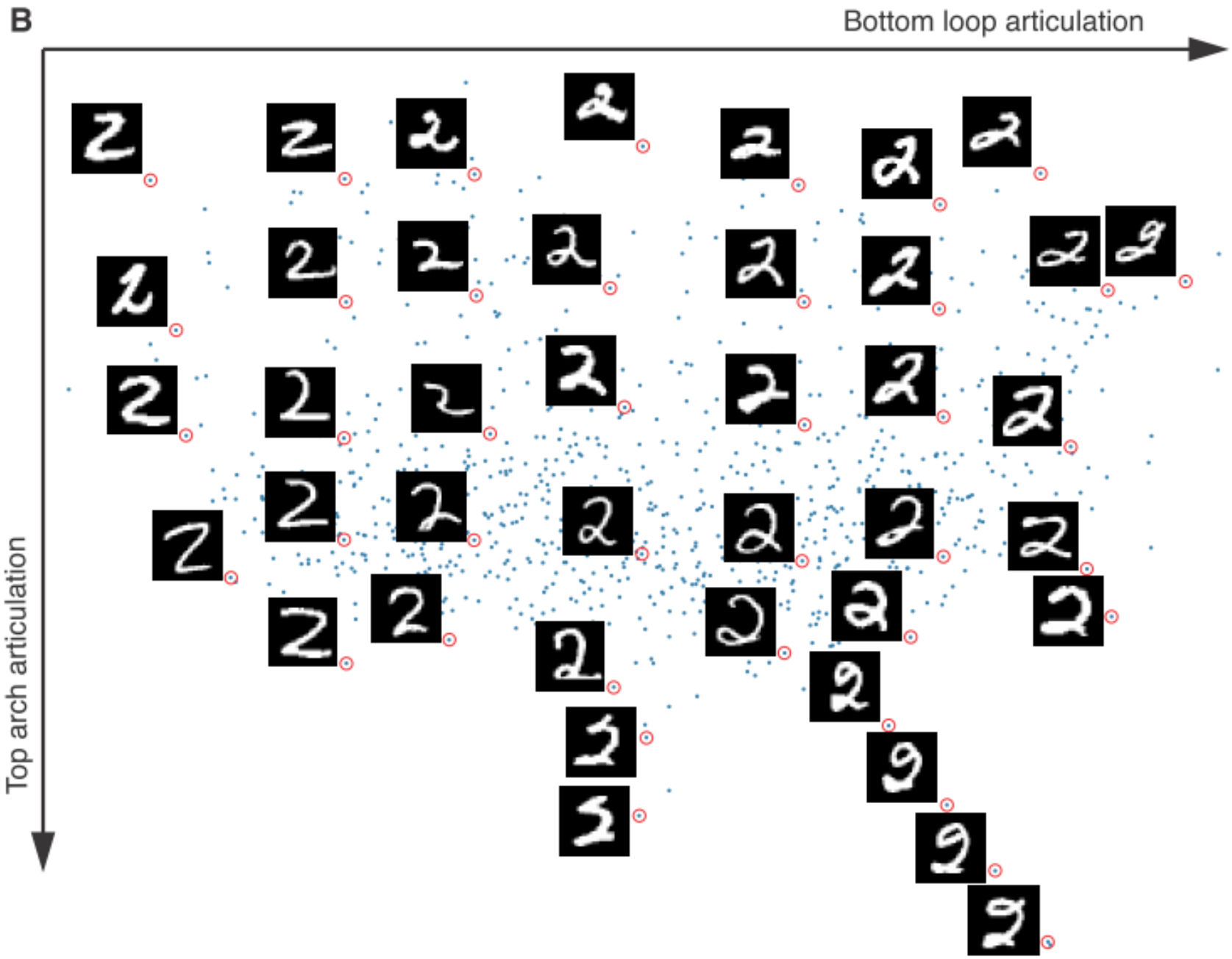
The Swiss-role Dataset



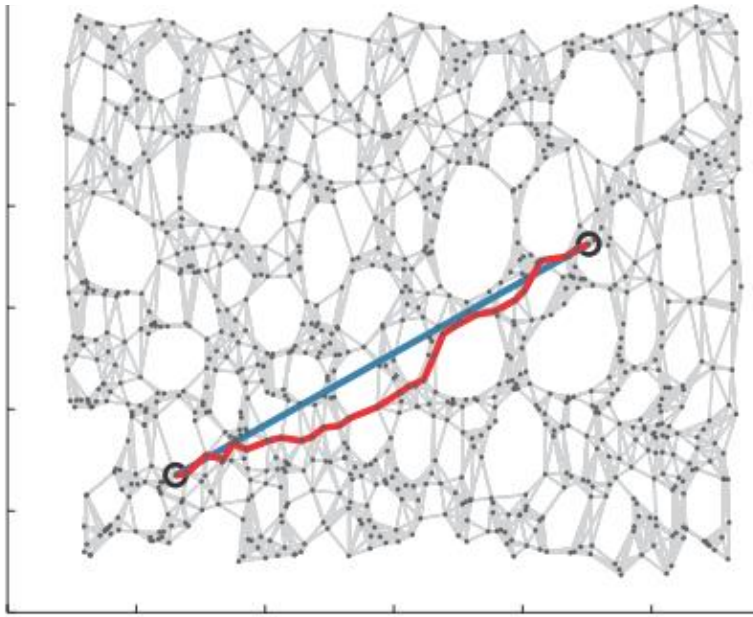
Isomap



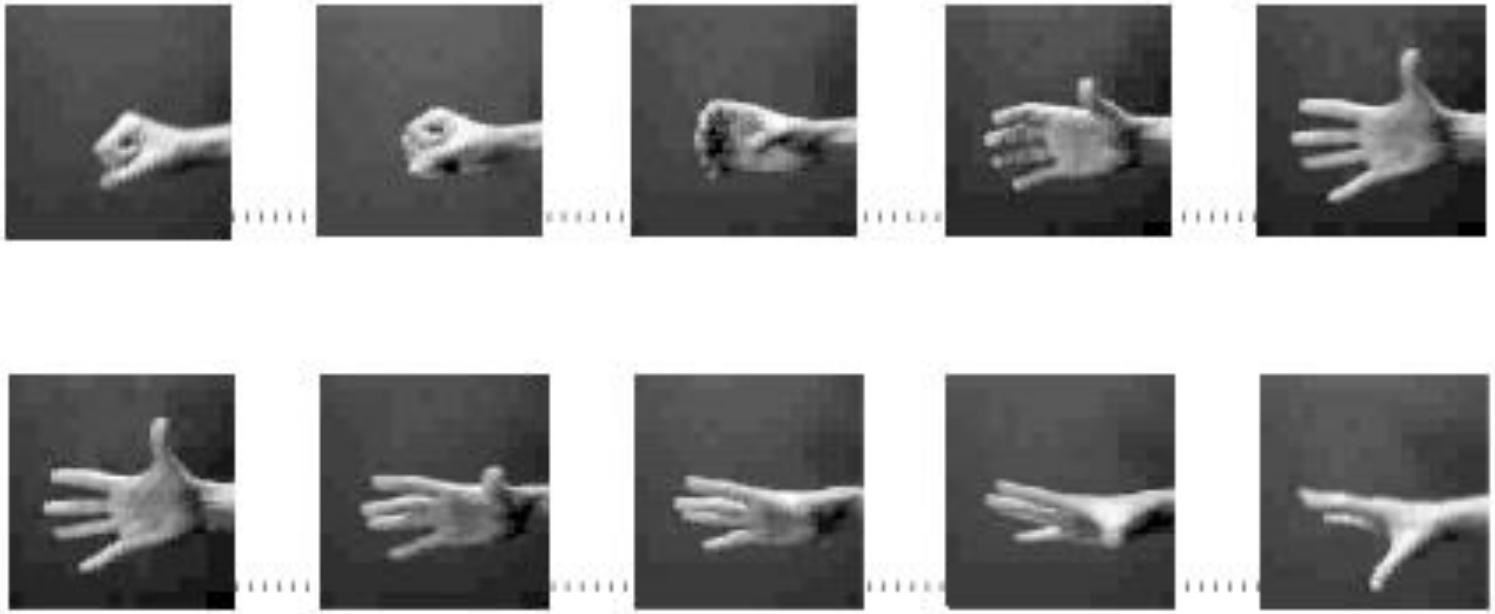

A
Up-down pose
Left-right pose
Lighting direction

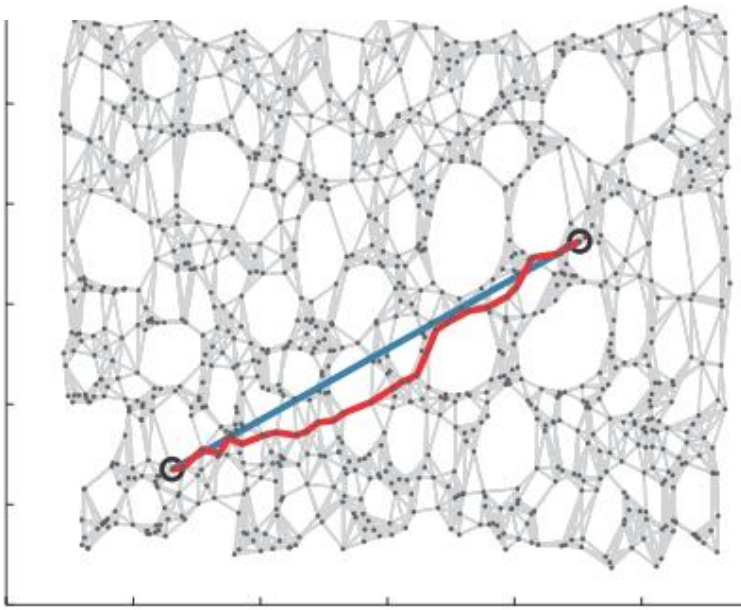

B
Bottom loop articulation
Top arch articulation

ISOMAP Interpolation

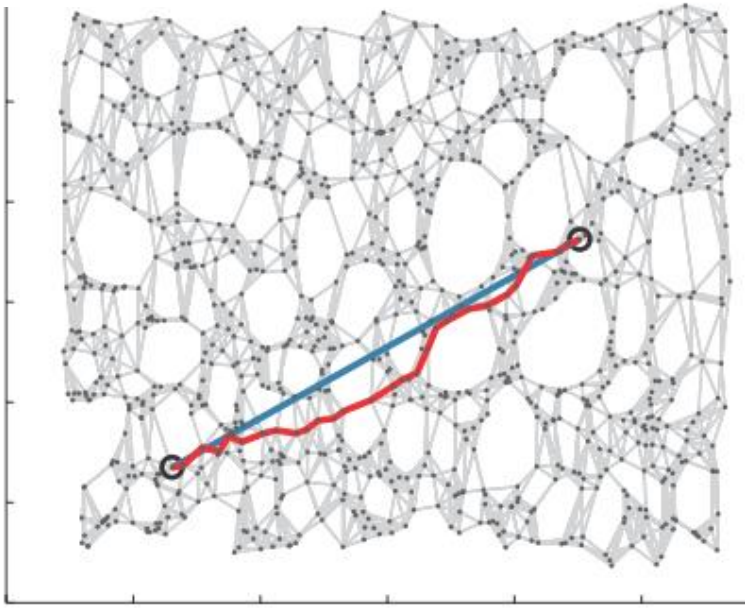


ISOMAP Interpolation





ISOMAP Interpolation



ISOMAP Summary

- Build graph from kNN or epsilon neighbors
- Run MDS

- Since MDS is slow, ISOMAP will be very slow.
- It has a parameter k or epsilon.

Maximum Variance Unfolding

K. Q. Weinberger and L. K. Saul.
Unsupervised learning of image manifolds by semidefinite programming.
International Journal of Computer Vision, Volume 70 Issue 1, October 2006,
Pages 77 - 90

Maximum Variance Unfolding

Step 1 Build a graph from kNN or epsilon neighbors.

Step 2 Given $x_1, \ldots, x_N$ find $y_1, \ldots, y_N$ such that

$\|x_i - x_j\| = \|y_i - y_j\|$ for all $(i,j) \in E$ neighborhood graph

and $var(y)$ is as large as possible.

Formally,

$$\max_y tr(cov(y))$$

s.t. $\|x_i - x_j\| = \|y_i - y_j\|$ for all $(i,j) \in E$ neighborhood graph

Here $tr(cov(y)) = \frac{1}{N}\sum_{i=1}^{N} \|y_i - \bar{y}\|^2$, where $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$

Maximum Variance Unfolding

$X = [x_1, \ldots, x_N] \in \mathbb{R}^{D \times N}$, $Y = [y_1, \ldots, y_N] \in \mathbb{R}^{d \times N}$

Let $P = X^T X$, $Q = Y^T Y \succeq 0$.

Our goal is to find $Q \succeq 0$. Then with PCA we can find $y_1, \ldots, y_N$ from Q.

Consider the constraint $\|x_i - x_j\| = \|y_i - y_j\|$

From this, we have $\|x_i - x_j\|^2 = \|y_i - y_j\|^2$

$$x_i^T x_i - 2x_i^T x_j + x_j^T x_j = y_i^T y_i - 2y_i^T y_j + y_j^T y_j$$

$$Q_{ii} - 2Q_{ij} + Q_{jj} = P_{ii} - 2P_{ij} + P_{jj}$$

Maximum Variance Unfolding

Consider the cost function:

$$\begin{aligned} cov(y) &= \frac{1}{N}\sum_{i=1}^{N}(y_i-\bar{y})(y_i-\bar{y})^T \\ &= \frac{1}{N}\sum_{i=1}^{N}y_i y_i^T - \bar{y}\bar{y}^T \\ &= \frac{1}{N}YY^T - \frac{1}{N^2}Y\mathbf{1}\mathbf{1}^TY^T \end{aligned}$$

$$\begin{aligned} tr(cov(y)) &= \frac{1}{N}tr(YY^T) - \frac{1}{N^2}tr(Y\mathbf{1}\mathbf{1}^TY^T) \\ &= \frac{1}{N}tr(YY^T) - \frac{1}{N^2}tr(Y^TY\mathbf{1}\mathbf{1}^T) \\ &= \frac{1}{N}tr(Q) - \frac{1}{N^2}tr(Q\mathbf{1}\mathbf{1}^T) \end{aligned}$$

Maximum Variance Unfolding

The final problem is a semi-definite problem (SDP) :

$$\max_{Q} \frac{1}{N} tr(Q) - \frac{1}{N^2} tr(Q\mathbf{1}\mathbf{1}^T)$$

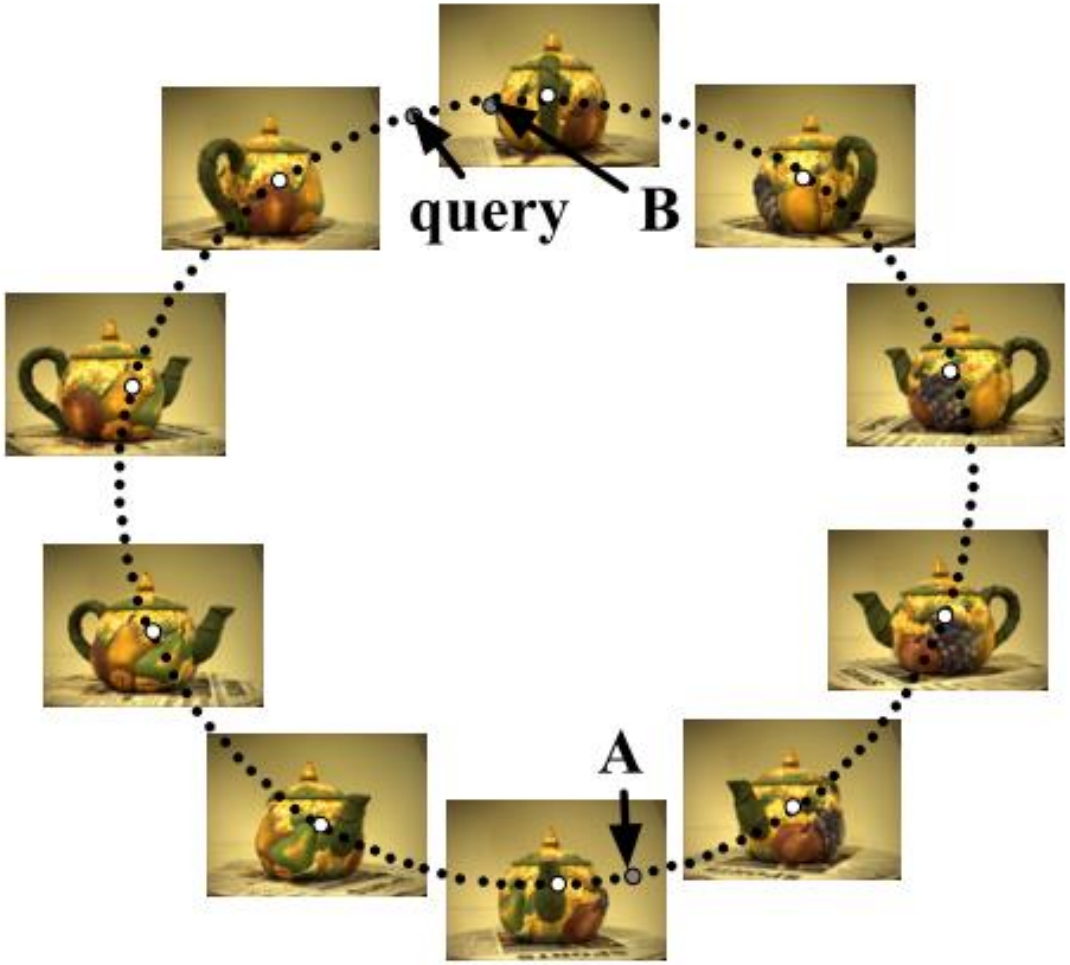
$$\text{s.t. } Q_{ii} - 2Q_{ij} + Q_{jj} = P_{ii} - 2P_{ij} + P_{jj} \text{ for all } (i,j) \in E,$$
$$Q \succeq 0$$

Maximum Variance Unfolding

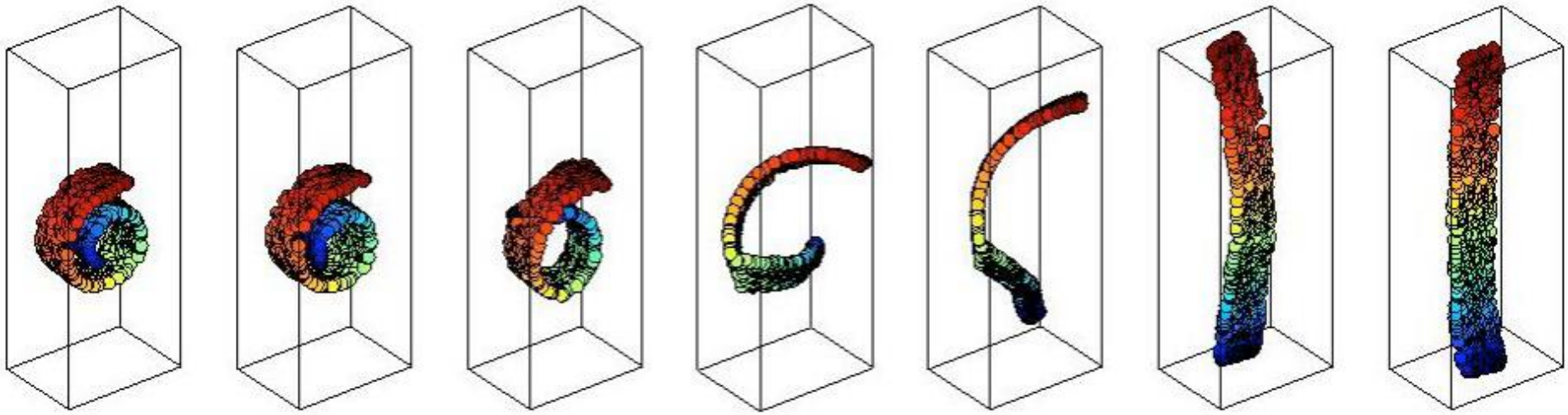
D= 76*101*3 (width, height, RGB)

d=2

N=400 images



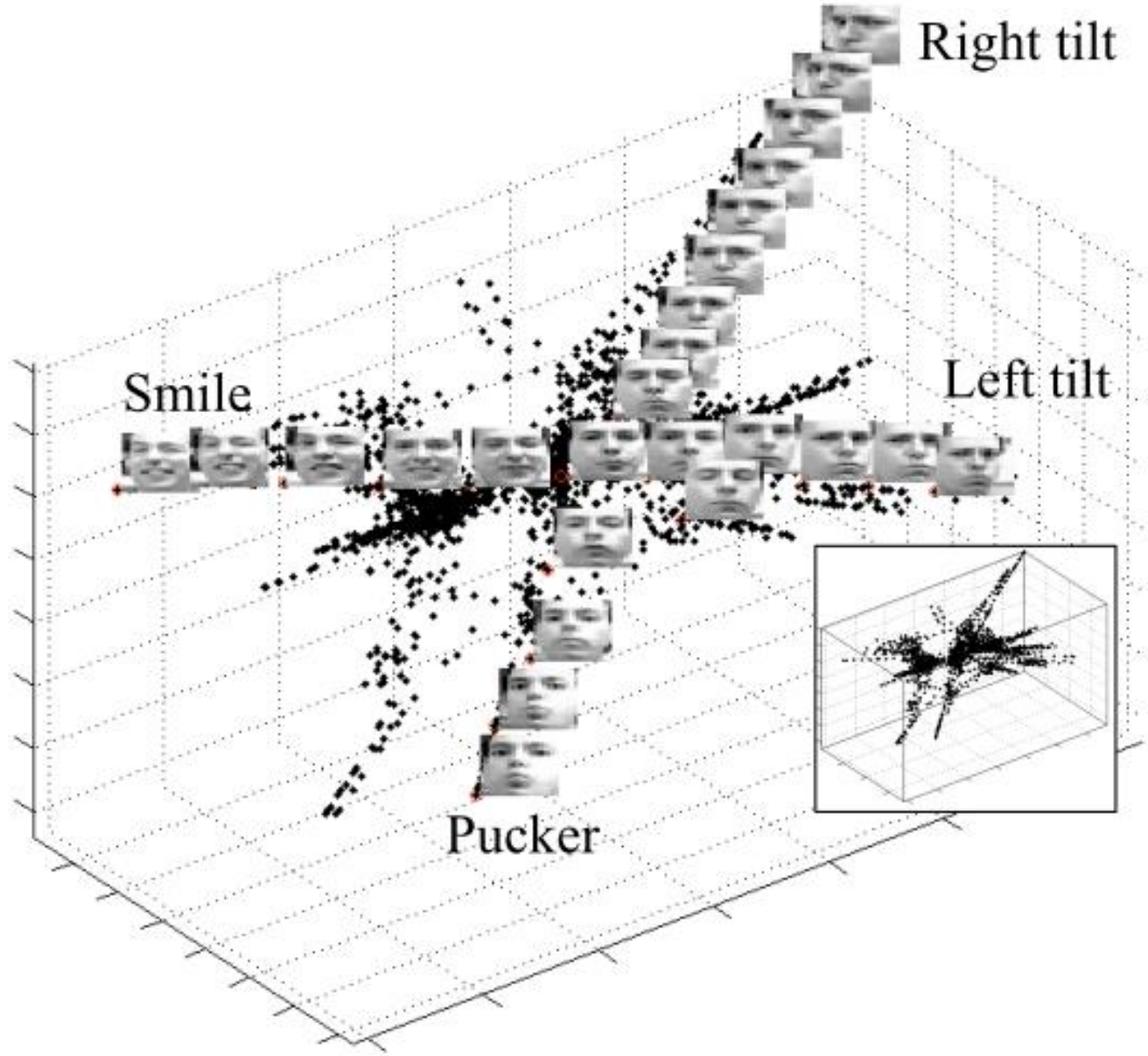
Maximum Variance Unfolding



Swiss roll "unfolded" by maximizing variance subject to constraints that preserve local distances and angles.

The middle snap-shots show various feasible (but non-optimal) intermediate solutions.

Maximum Variance Unfolding



Laplacian Eigenmap

M. Belkin and P. Niyogi. "Laplacian eigenmaps for dimensionality reduction and data representation,"
Neural Comput., 15(6):1373–1396, 2003.

Laplacian Eigenmap

Data: $X = [x_1, \ldots, x_N] \in \mathbb{R}^{D \times N}$

Step 1. Build graph from kNN or epsilon neighbors

Step 2. Choose weights:

$$W_{ij} = \exp(-\tfrac{1}{t}\|x_i - x_j\|^2) \text{ if } (i,j) \in E$$

$$W_{ij} = 0 \text{ Otherwise}$$

Special case:

$$t = \infty, \text{ then } W_{ij} = 1 \text{ if } (i,j) \in E$$

Observation:

$$W \text{ is symmetric: } W_{ij} = W_{ji} \ \forall i,j$$

Laplacian Eigenmap

Step 3. Assume the graph is connected, otherwise proceed with Step 3 for each connected component:

$$D_{ii} = \sum_{j=1}^{N} W_{ij} \quad D \in \mathbb{R}^{N \times N} \text{ diagonal matrix}$$

$$L = D - W \in \mathbb{R}^{N \times N} \text{ Laplacian matrix}$$

Lemma:

L is symmetric, positive semi-definite matrix.

Laplacian Eigenmap

Step 4. Solve the eigenvector problem:

$$Lf = \lambda Df \quad f \in \mathbb{R}^N$$

The first m+1 smallest eigenvalues:

$$Lf_0 = \lambda_0 Df_0 \quad 0 = \lambda_0,\ f_0 = [1, \ldots, 1]^T \in \mathbb{R}^N$$

$$Lf_1 = \lambda_1 Df_1$$

$$\vdots$$

$$Lf_m = \lambda_m Df_m \quad 0 = \lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_m$$

The embedding:

$$\mathbb{R}^N \ni x_i \rightarrow [f_1(i), \ldots f_m(i)]^T \in \mathbb{R}^m$$

Laplacian Eigenmap (Explanation)

Let us embed the neighborhood graph to 1 dim first.

A reasonable cost function is:

$$\min_{y_1,\dots y_N} \sum_{i,j=1}^{N} (y_i - y_j)^2 W_{ij}$$

subject to appropriate constraints to avoid y=0.

Lemma

$$\sum_{i,j=1}^{N} (y_i - y_j)^2 W_{ij} = 2y^T L y$$

$D_{ii} = \sum_{j=1}^{N} W_{ij}$

$L = D - W \in \mathbb{R}^{N \times N}$ Laplacian matrix

Proof:

$$\begin{aligned}
\sum_{i,j=1}^{N} (y_i - y_j)^2 W_{ij} &= \sum_{i,j=1}^{N} (y_i^2 + y_j^2 - 2y_i y_j) W_{ij} \\
&= \sum_i y_i^2 D_{ii} + \sum_j y_j^2 D_{jj} - 2 \sum_{i,j} y_i y_j W_{ij} \\
&= 2y^T L y
\end{aligned}$$

Laplacian Eigenmap (Explanation)

Therefore, our minimization problem is

$$\min_{y=[y_1,\dots y_N]^T} y^T L y$$

Subject to:

$y^T D y = 1$ to fix the scaling.

$y^T D \mathbf{1} = 0$ to avoid the trivial $y = [1, \dots, 1]^T$ solution.

Embedding the neighborhood graph to d dimension:

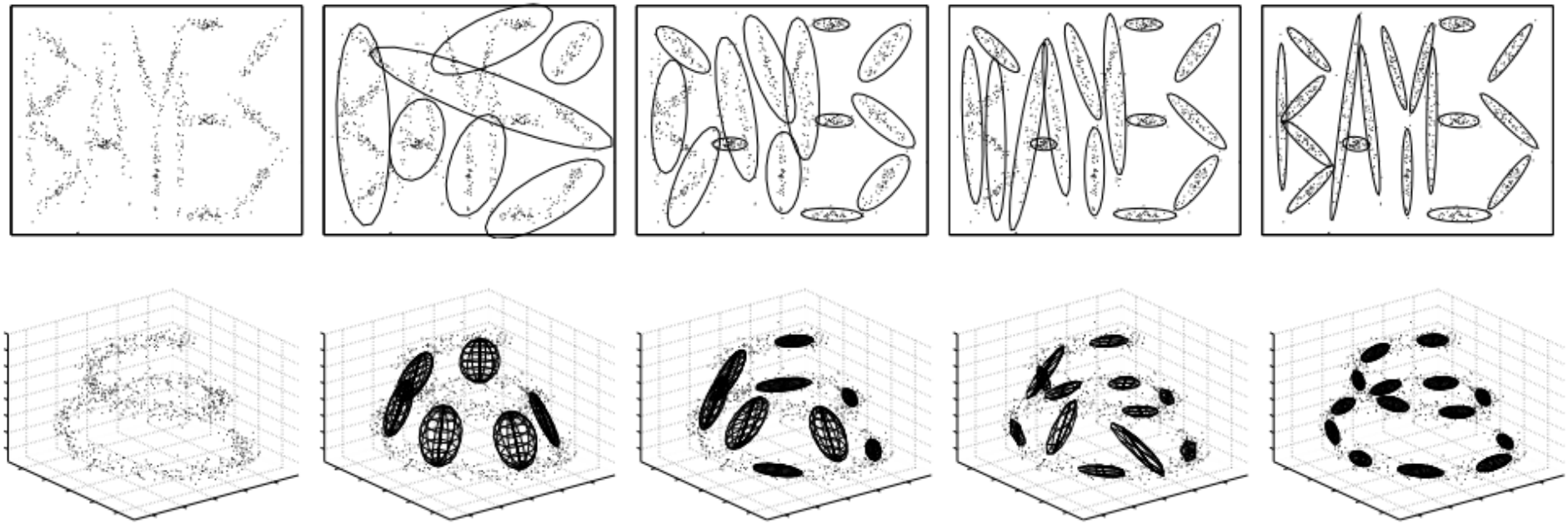
$$\min_{Y^T=[y_1,\dots y_N]\in\mathbb{R}^{d\times N}} tr(Y^T L Y)$$

Subject to: $Y^T D Y = I$

Solution: $LY = \lambda DY$

Variational Variational Inference for Bayesian Mixtures of Factor Analysers

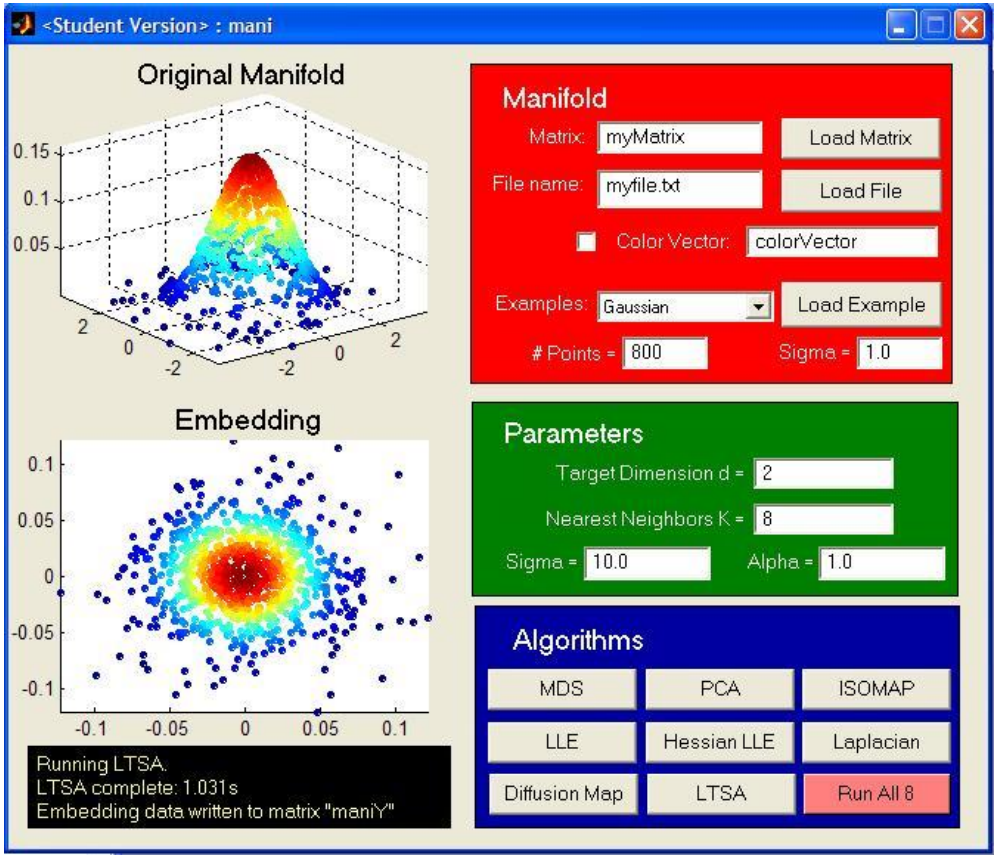
Zoubin Ghahramani, Matthew J. Beal, NIPS 1999



MANI Matlab demo

Todd Wittman: MANIfold learning demonstration GUI
Contains a couple of methods and examples.

http://www.math.ucla.edu/~wittman/mani



The following results are taken from Todd Wittman

How do we compare the methods?

- Speed
- Manifold Geometry
- Non-convexity
- Curvature
- Corners

- Noise
- Non-uniform Sampling
- Sparse Data
- Clustering

- High-Dimensional Data: *Can the method process image manifolds?*

- Sensitivity to Parameters
  - K Nearest Neighbors: *Isomap, LLE, Hessian, Laplacian, KNN Diffusion*
  - Sigma: *Diffusion Map, KNN Diffusion*

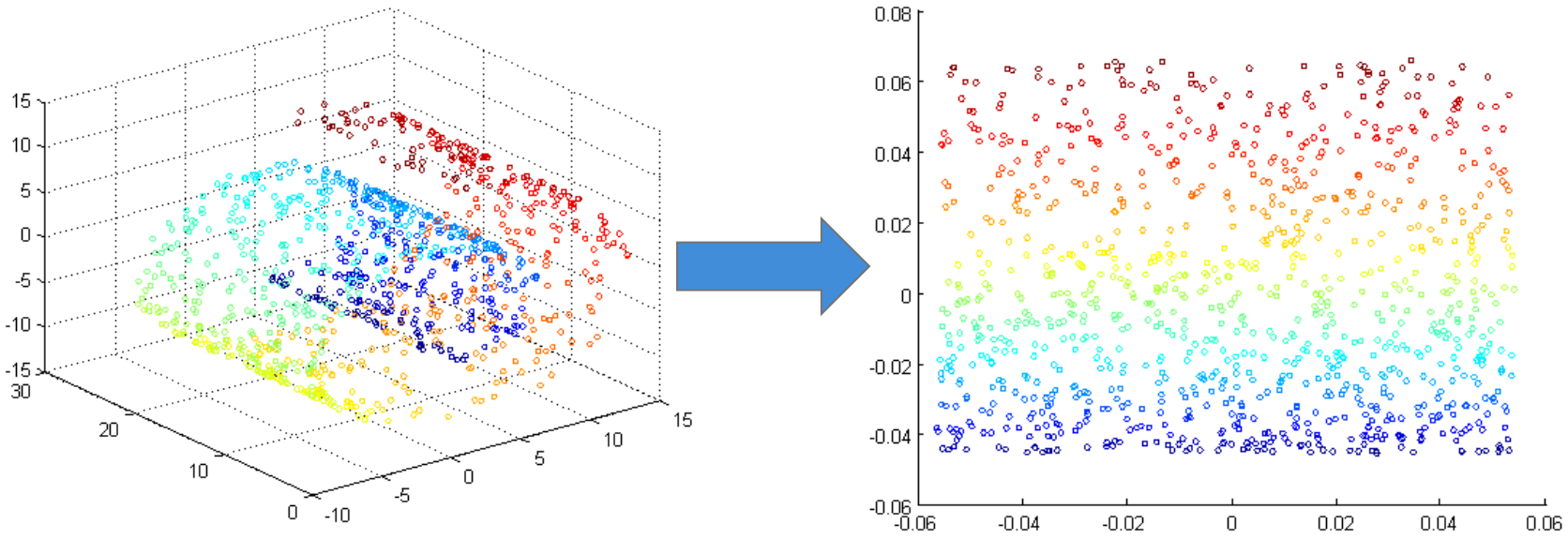
Testing Examples

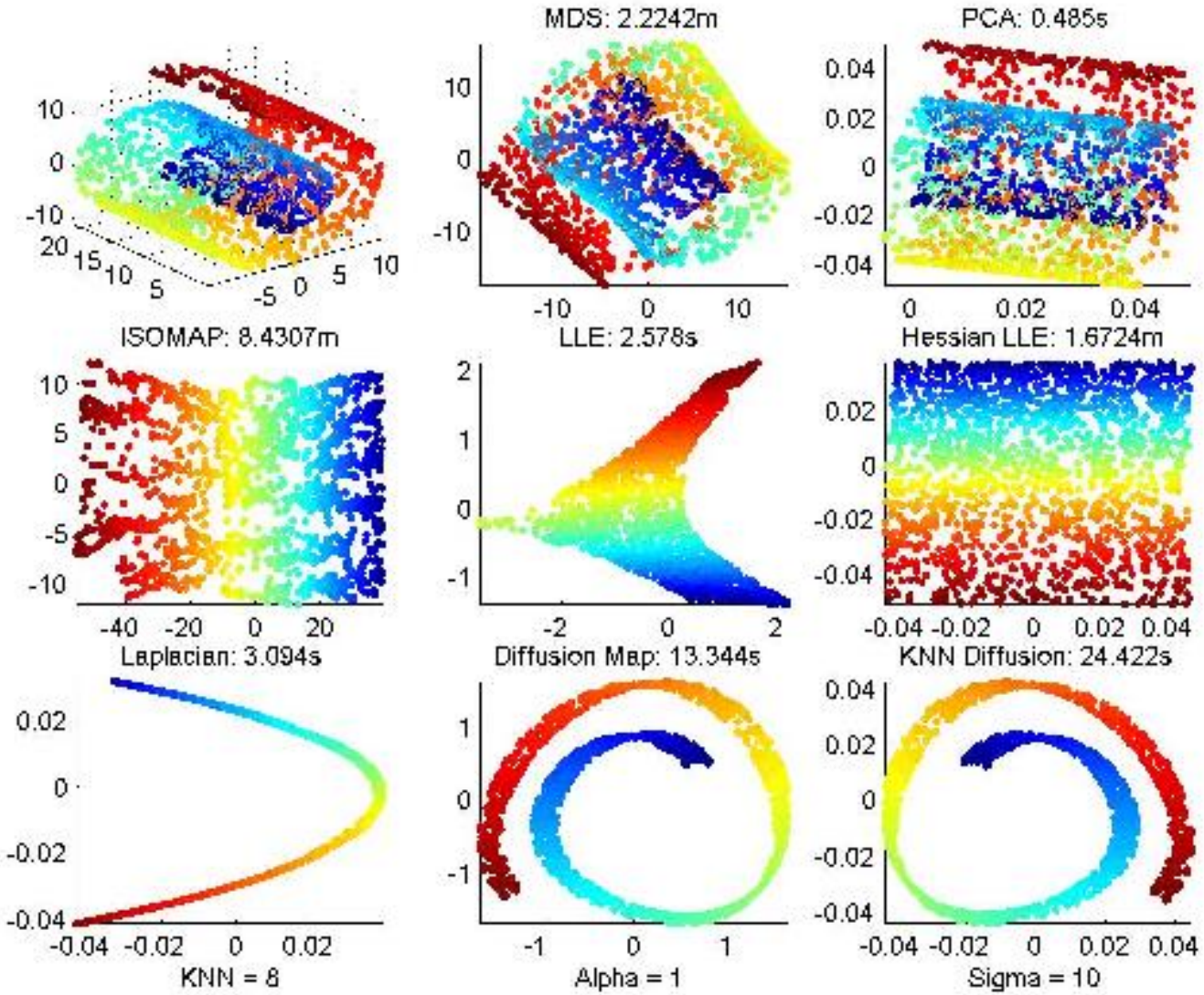
- ❑ Swiss Roll
- ❑ Swiss Hole
- ❑ Punctured Sphere
- ❑ Corner Planes
- ❑ 3D Clusters
- ❑ Twin Peaks
- ❑ Toroidal Helix
- ❑ Gaussian
- ❑ Occluded Disks

We'll compare the speed and sensitivity to parameters throughout.

Manifold Geometry

First, let's try to unroll the Swiss Roll.
We should see a plane.

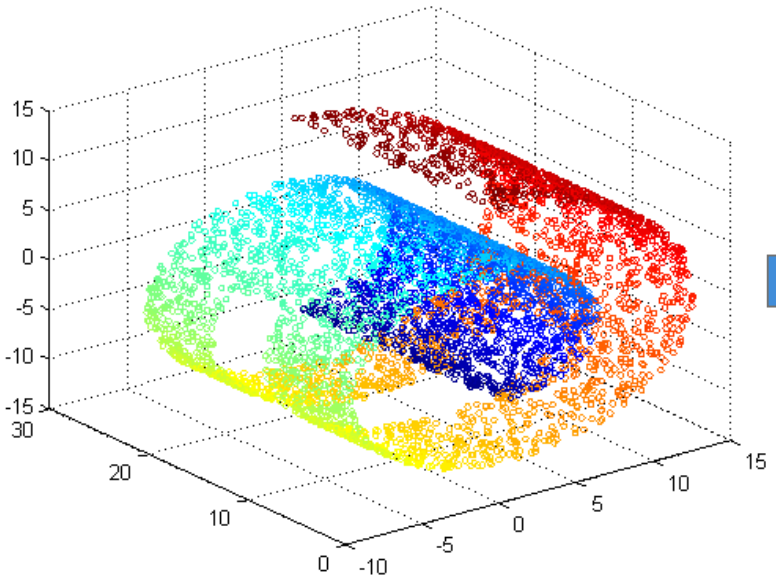


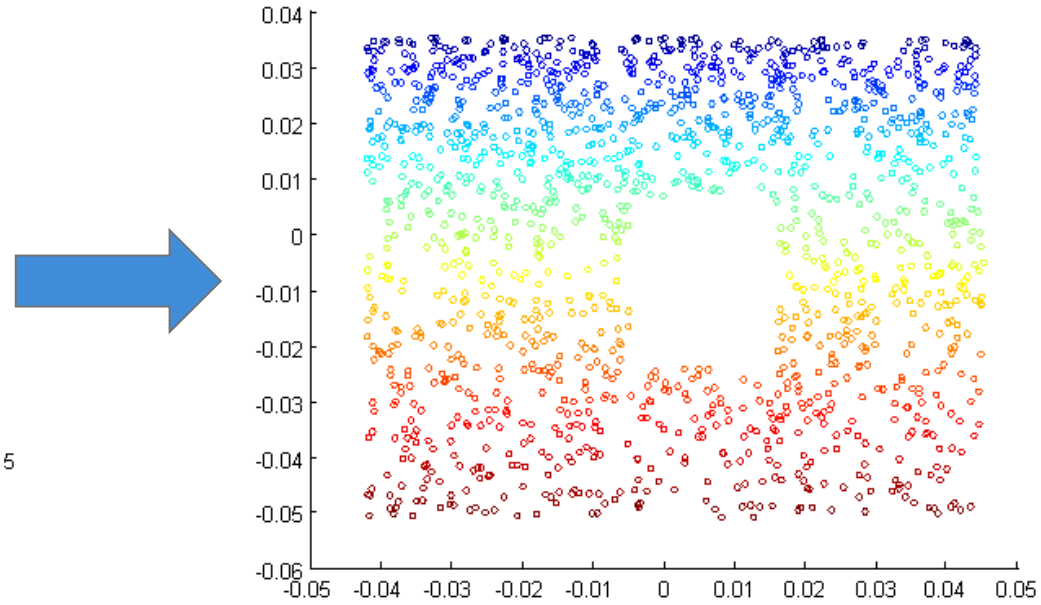


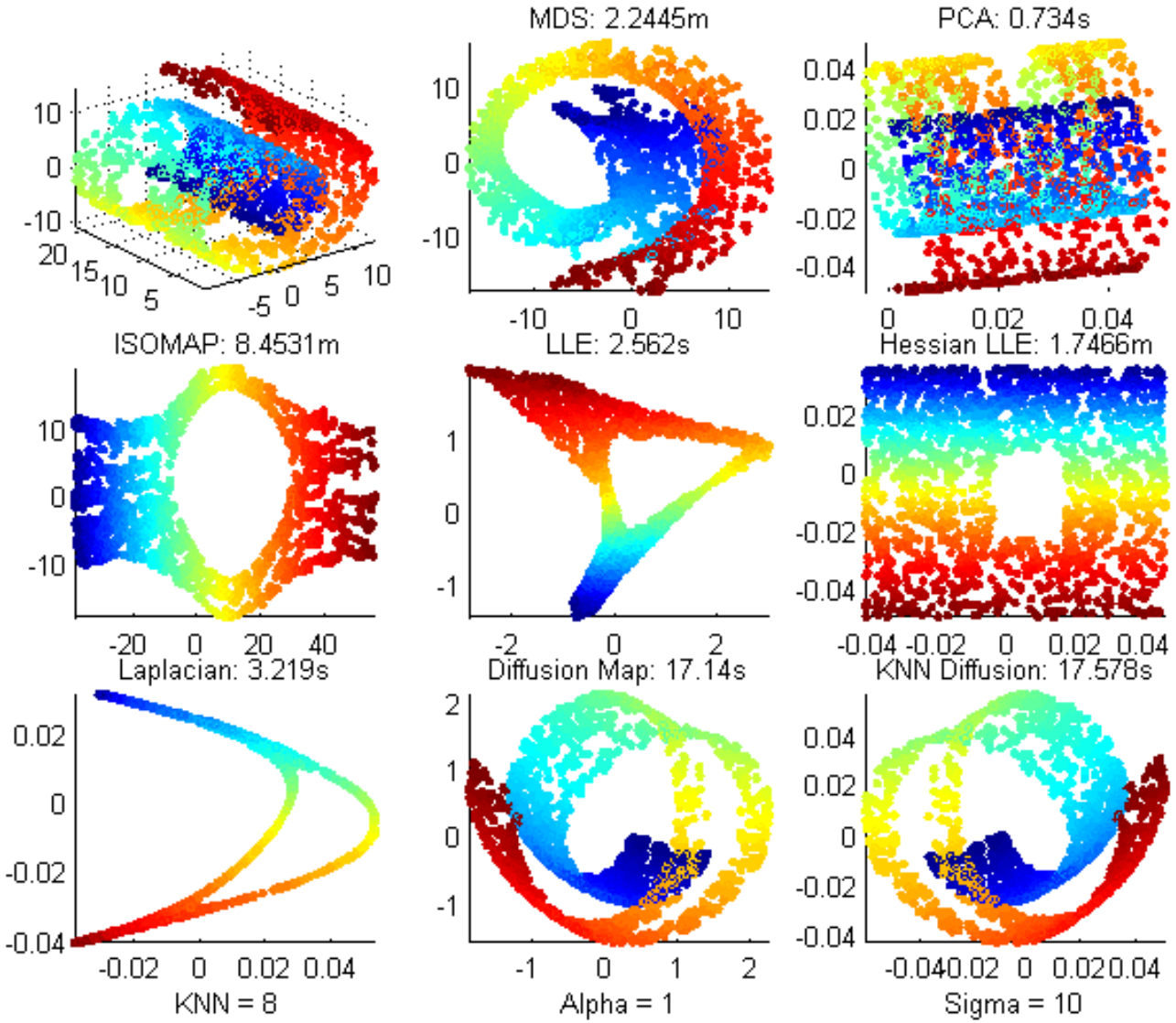
Hessian LLE is pretty slow, MDS is very slow, and ISOMAP is extremely slow.
MDS and PCA can't unroll the Swiss Roll.
Laplacian can't handle this data.
Diffusion Maps could not unroll Swiss Roll for any value of Sigma.

Non-Convexity

Can we handle a data set with a hole?
Swiss Hole: Can we still unroll the Swiss Roll when it has a hole in the middle?



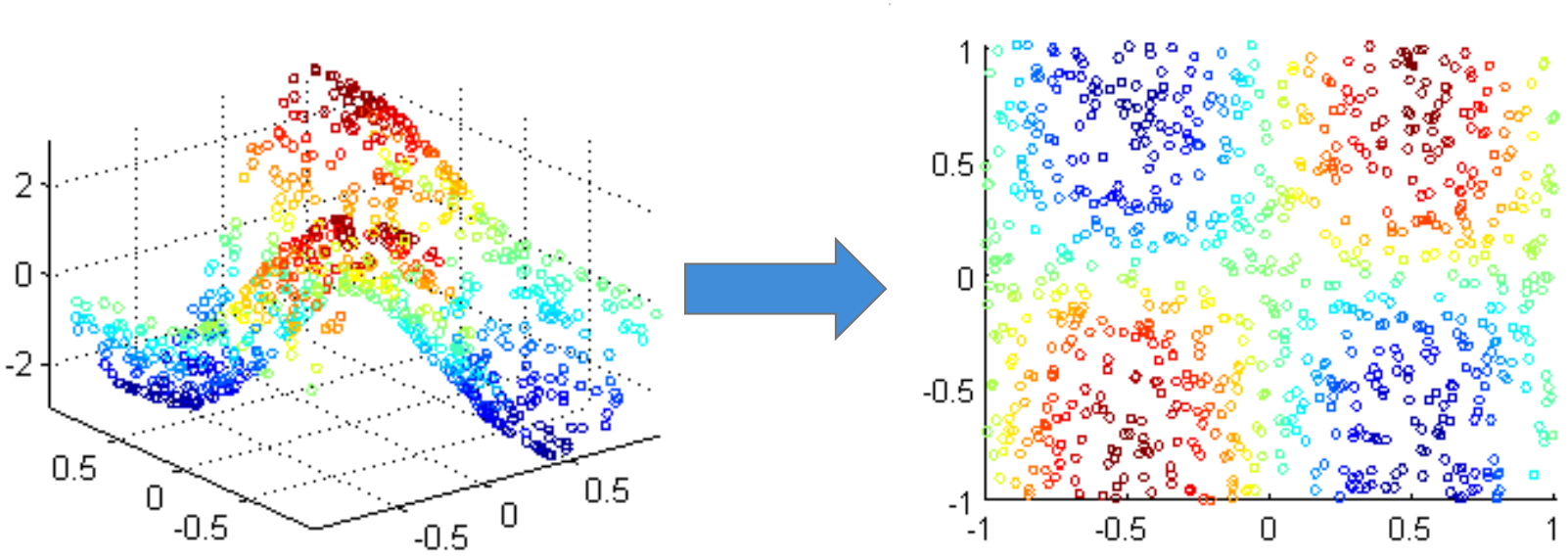


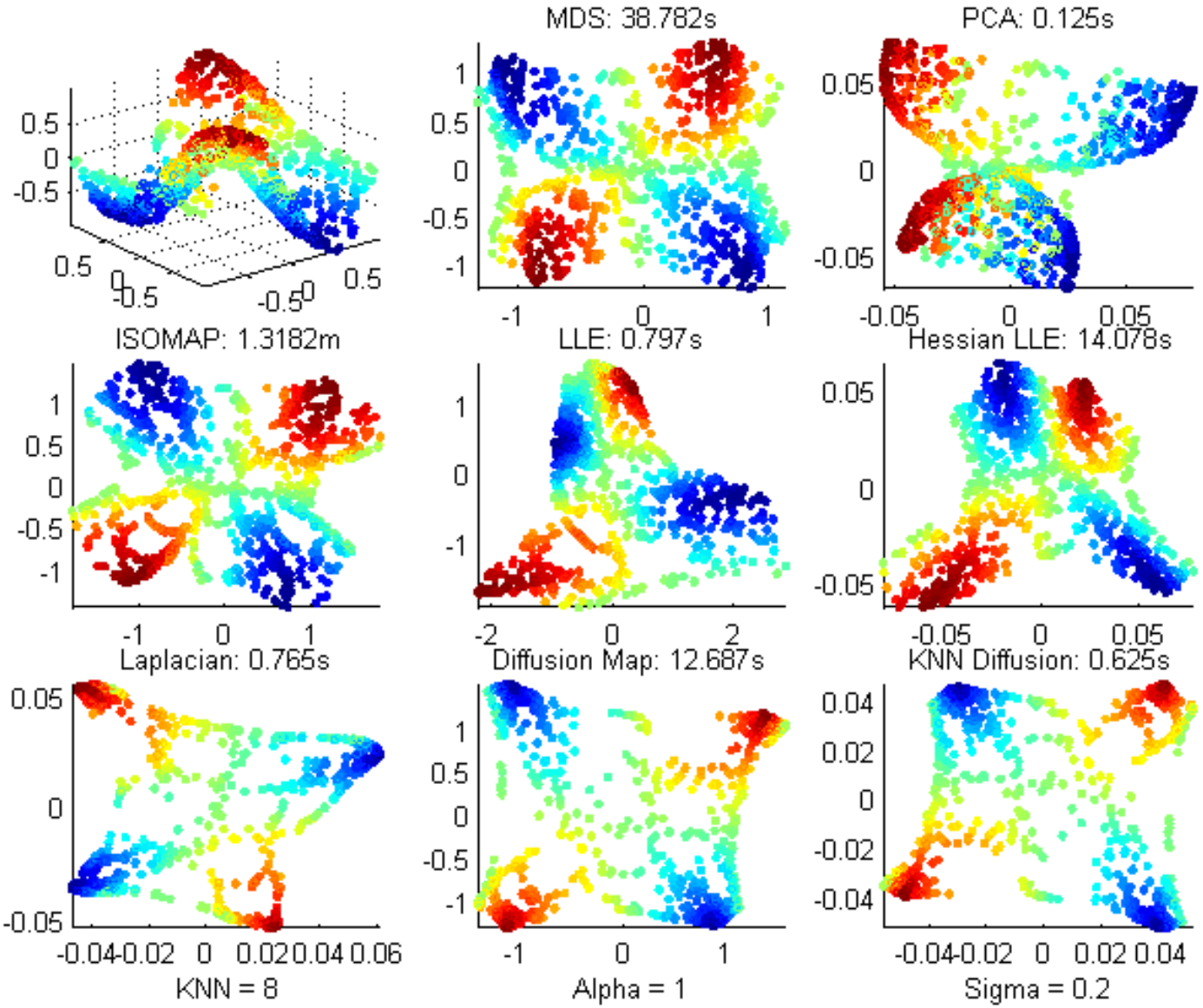


Only Hessian LLE can handle non-convexity.
ISOMAP, LLE, and Laplacian find the hole but the set is distorted.

Manifold Geometry

Twin Peaks: fold up the corners of a plane.

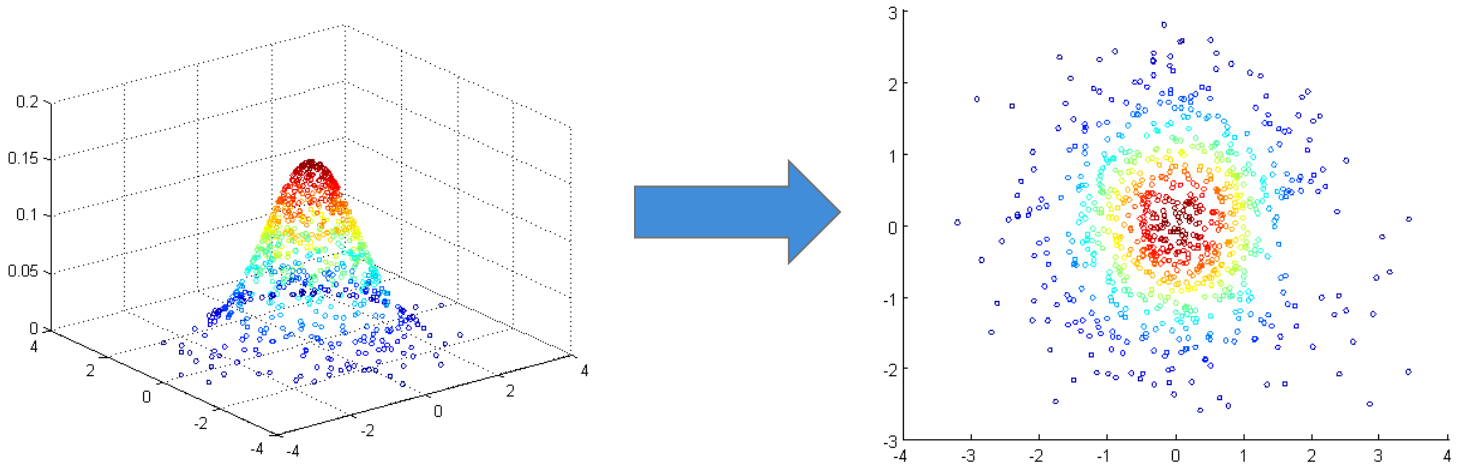


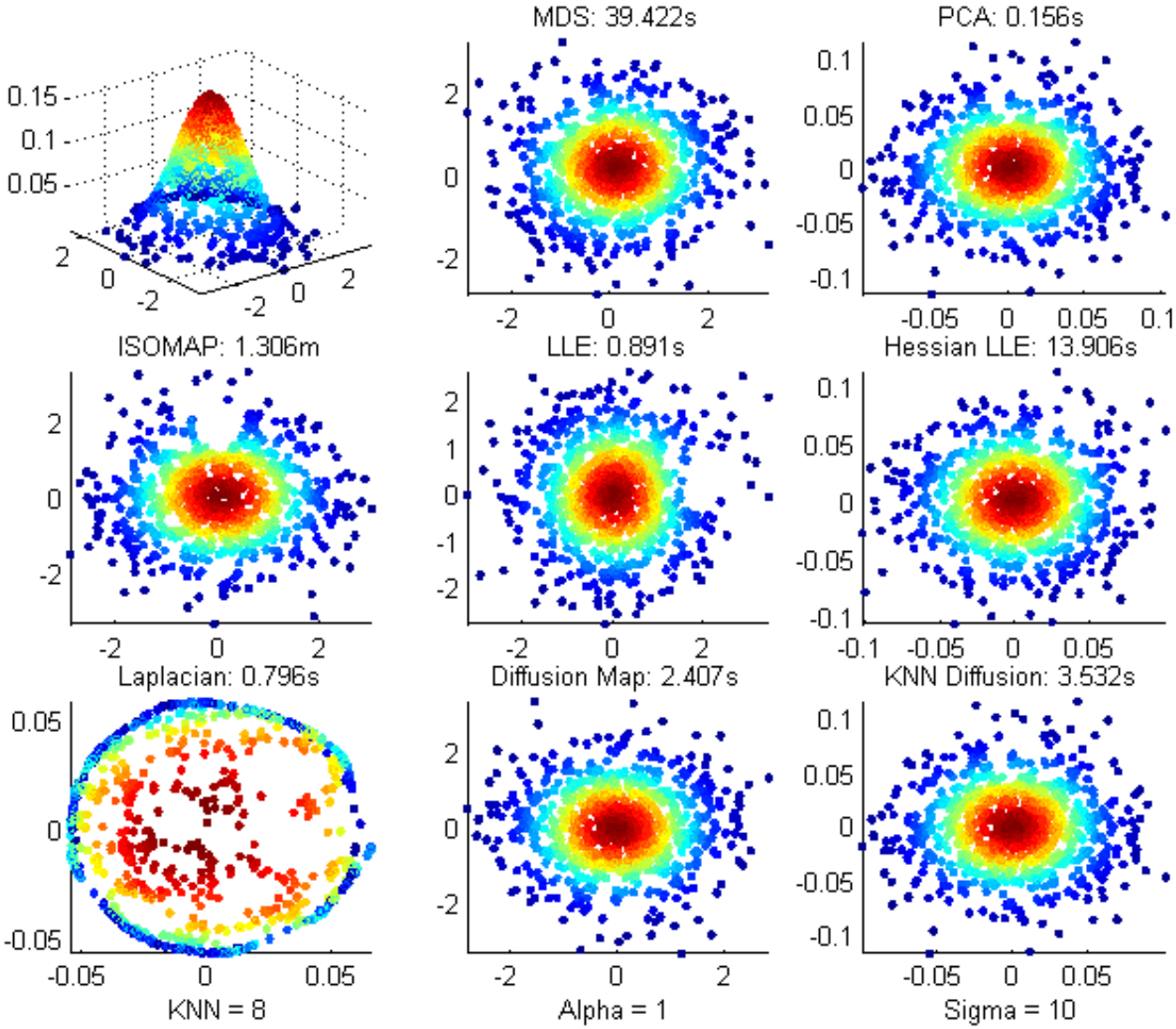


PCA, LLE, and Hessian LLE distort the mapping the most.

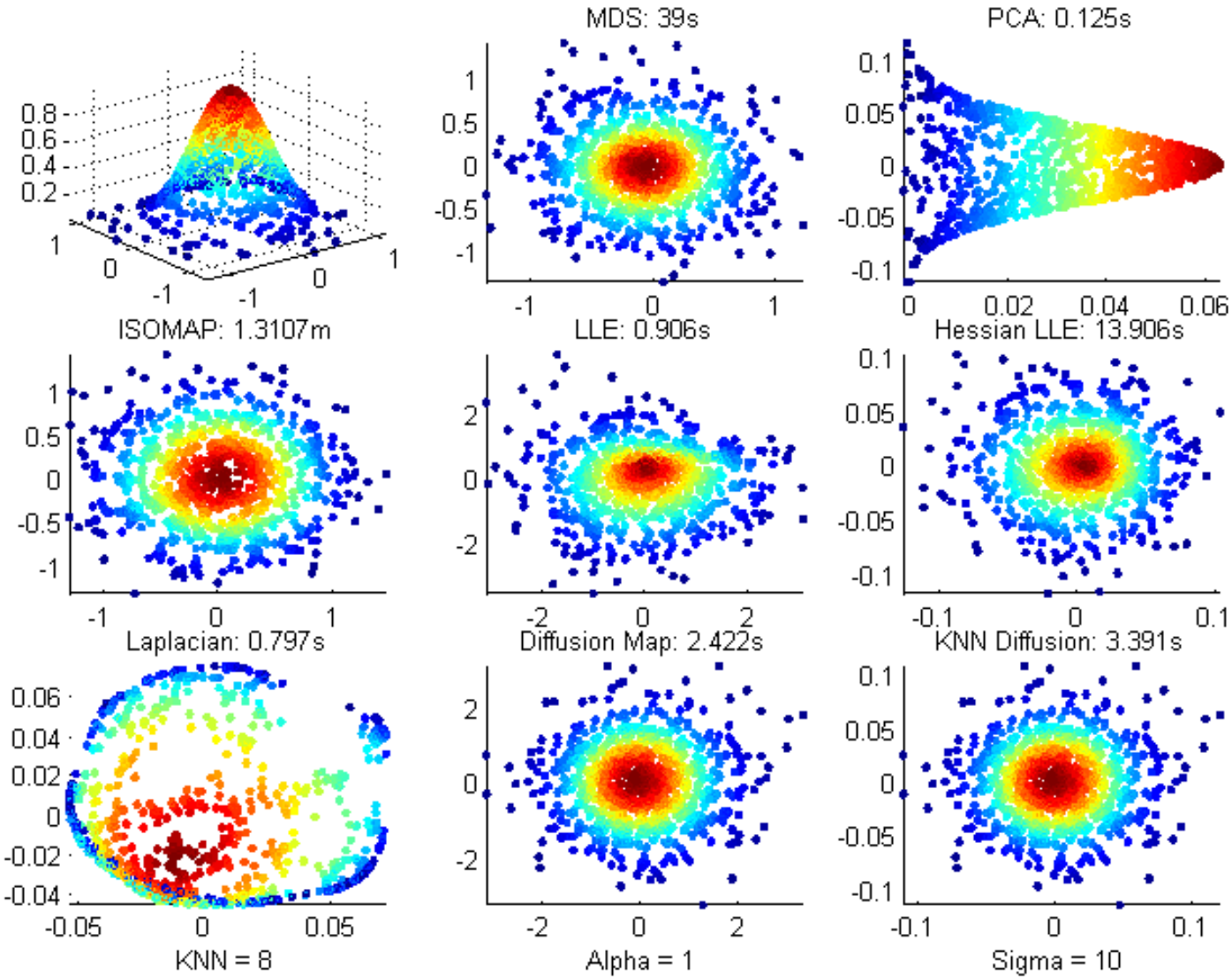
Curvature & Non-uniform Sampling

Gaussian: Randomly sample from a Gaussian distribution.
We increase the curvature by decreasing the standard deviation.
Coloring on the z-axis, we should map to concentric circles.

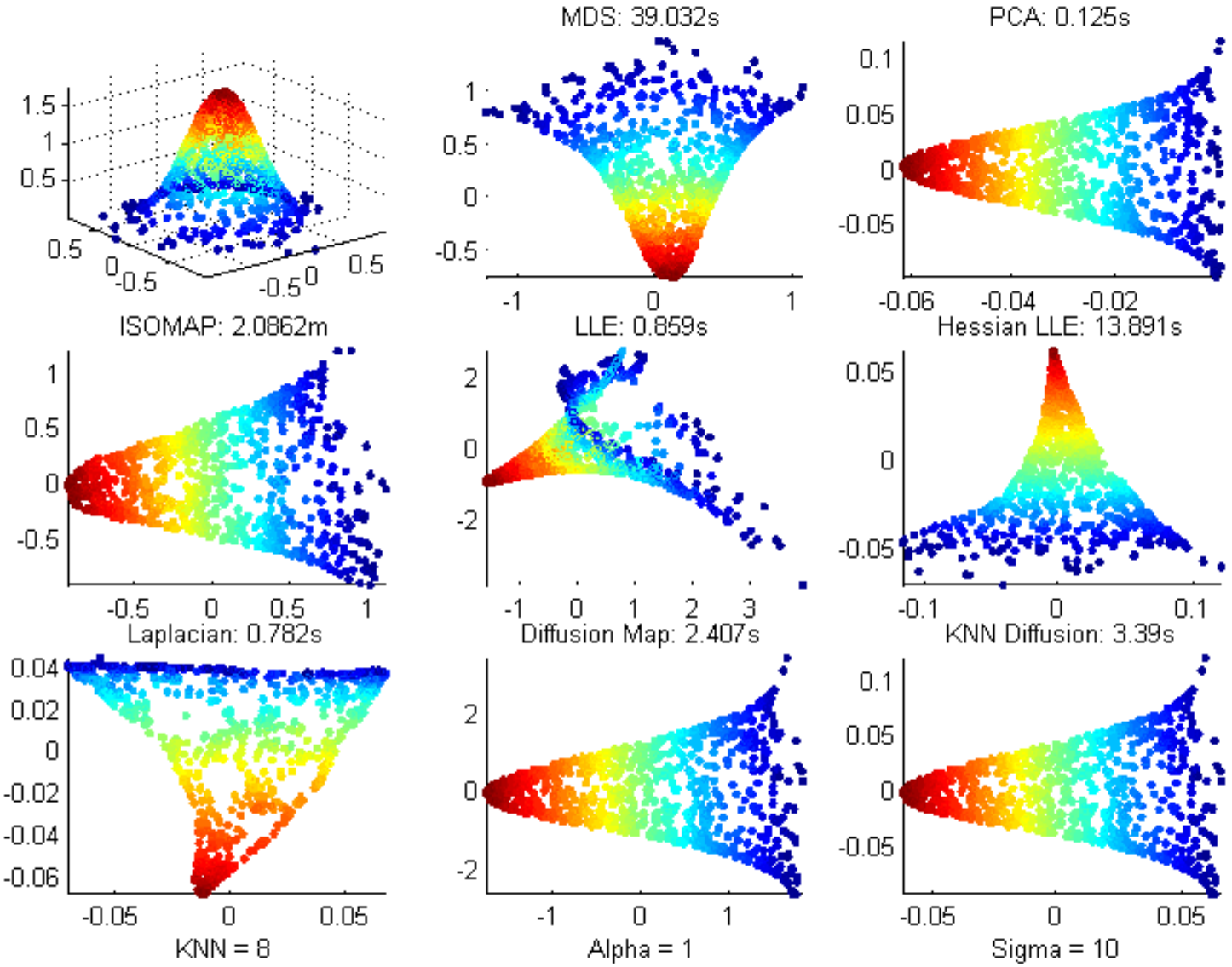




For std = 1 (low curvature), MDS and PCA can project accurately. Laplacian Eigenmap cannot handle the change in sampling.



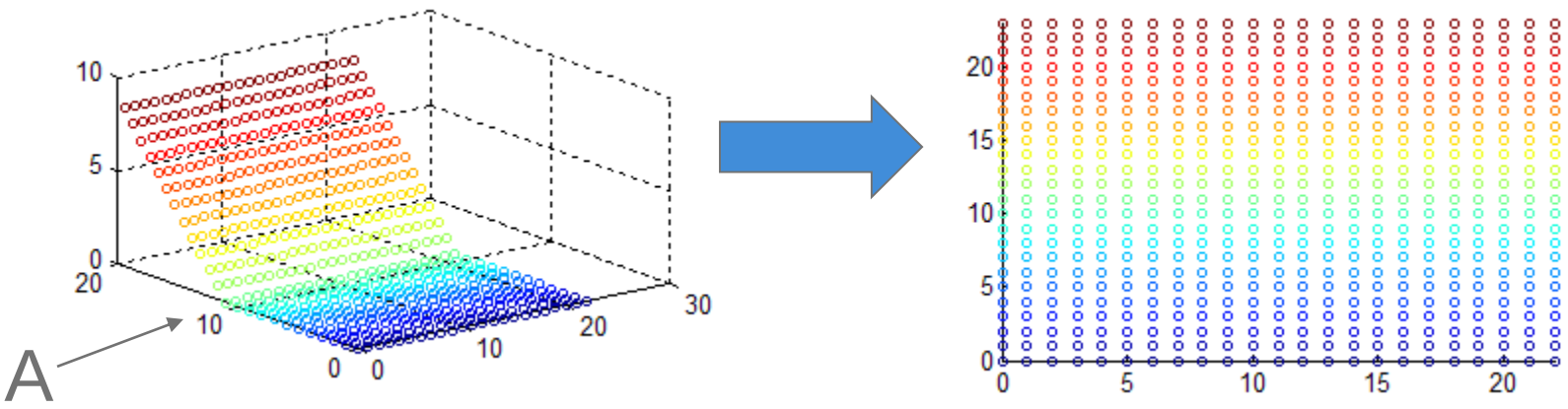
For std = 0.4 (higher curvature), PCA projects from the side rather than top-down. Laplacian looks even worse.



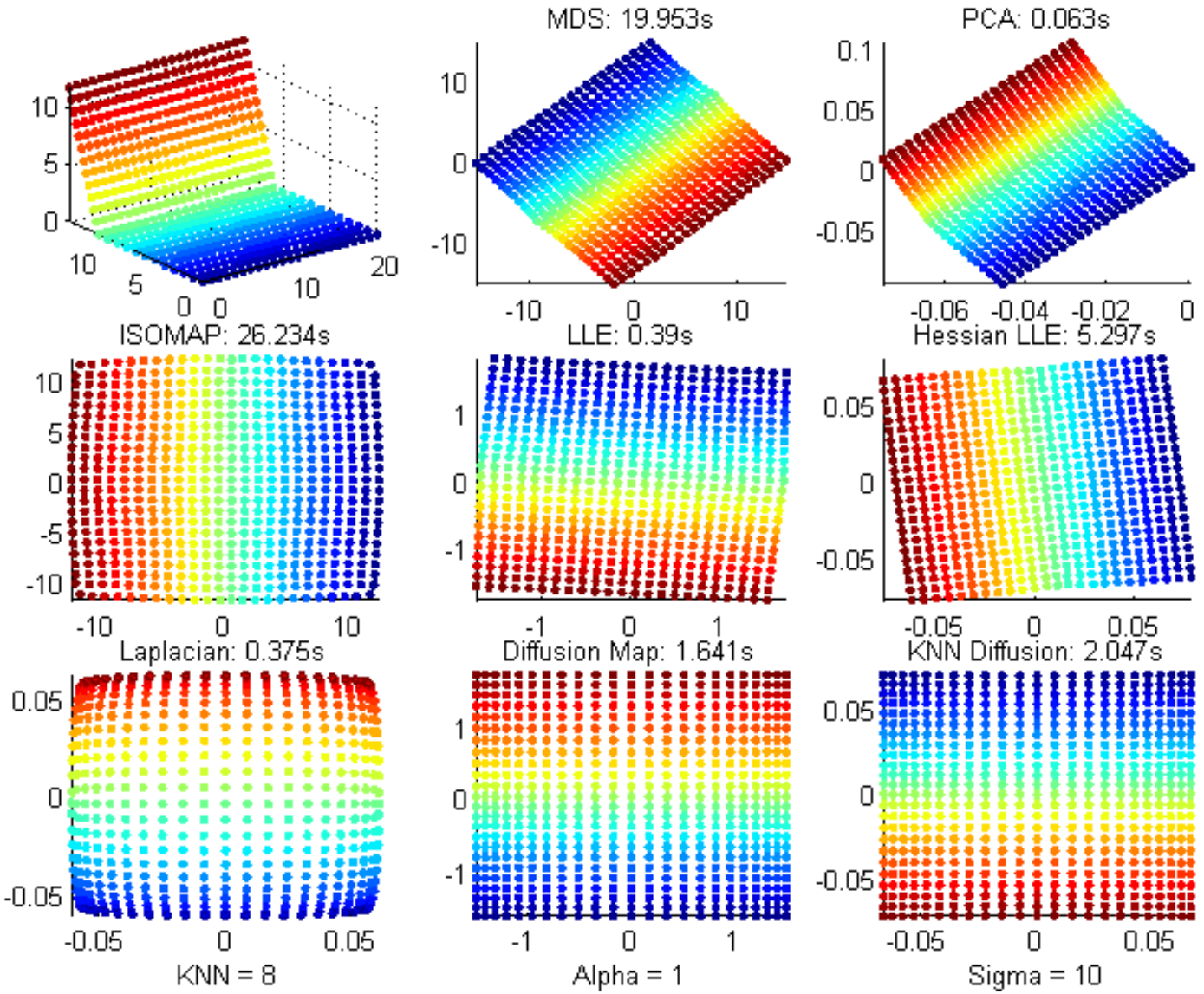
For std = 0.3 (high curvature), none of the methods can project correctly.

Corners

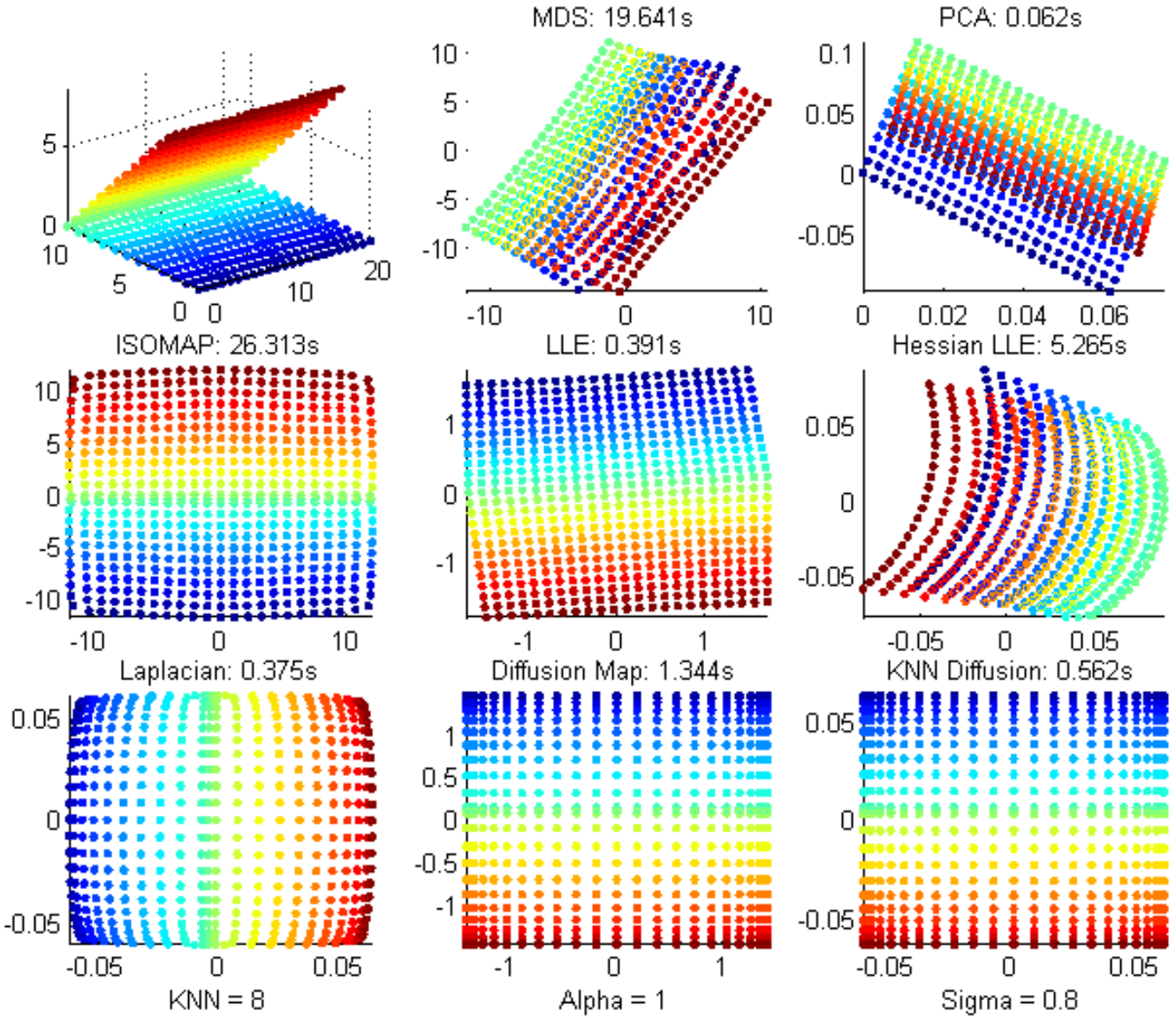
Corner Planes: We bend a plane with a lift angle A.
We want to bend it back down to a plane.



If A > 90, we might see the data points written on top of each other.



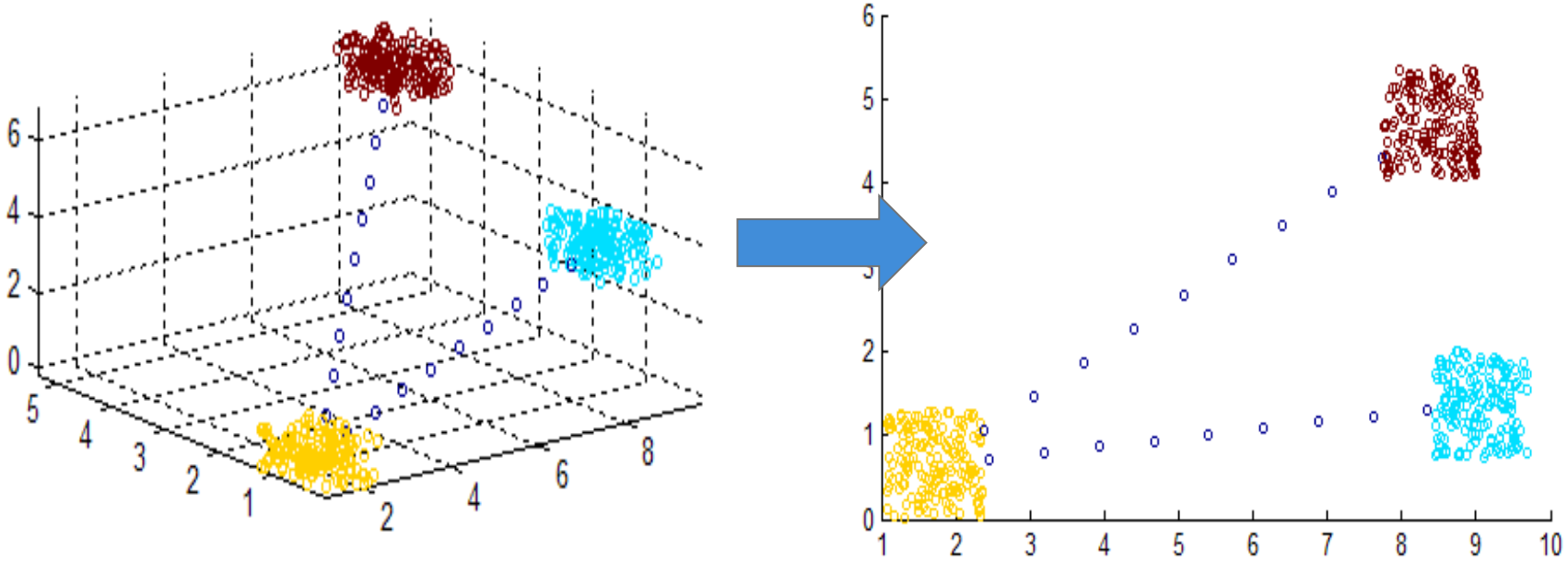
For angle A=75, we see some disortions in PCA and Laplacian.

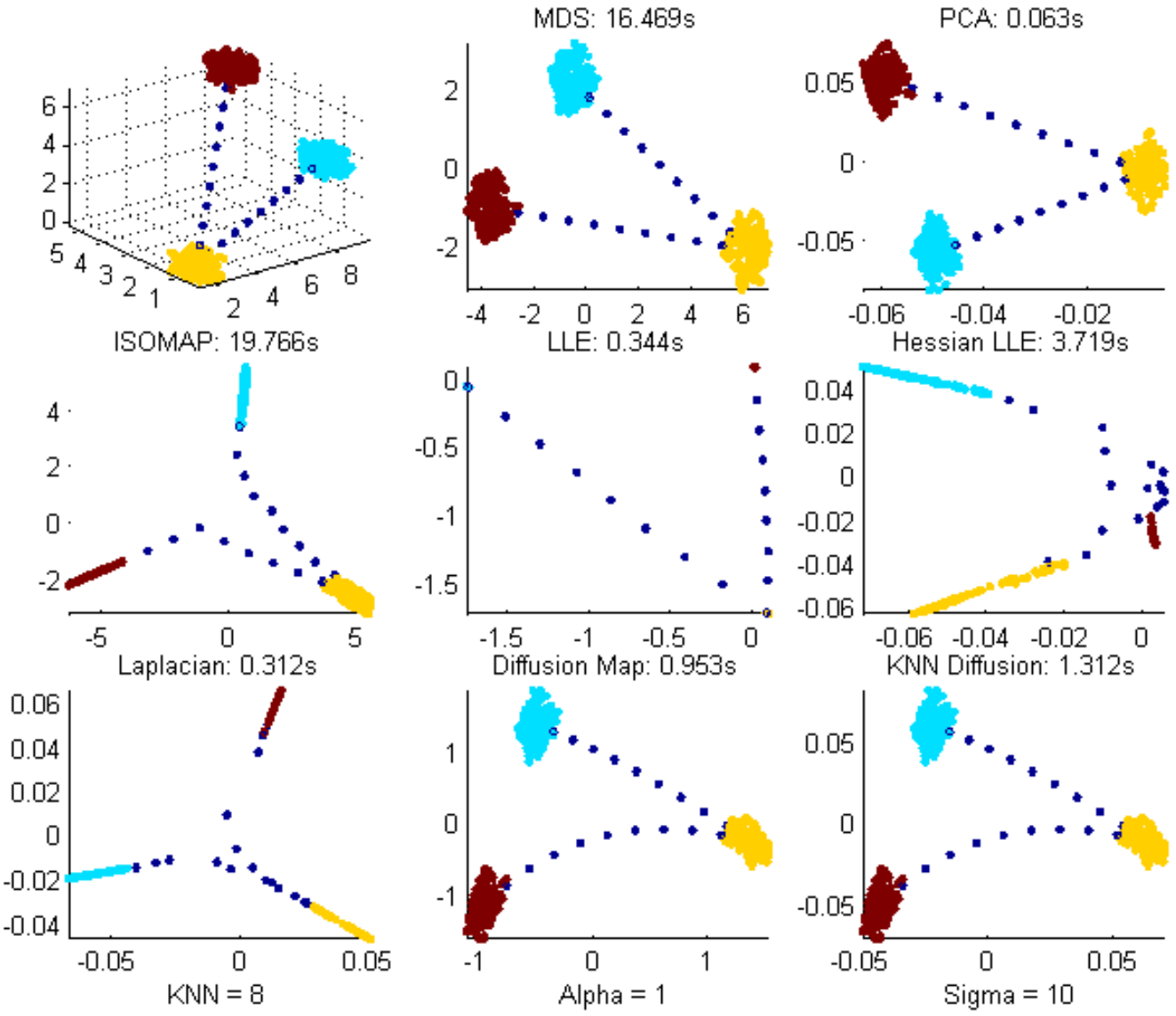


For A = 135, MDS, PCA, and Hessian LLE overwrite the data points.
Diffusion Maps work very well for Sigma < 1.
LLE handles corners surprisingly well.

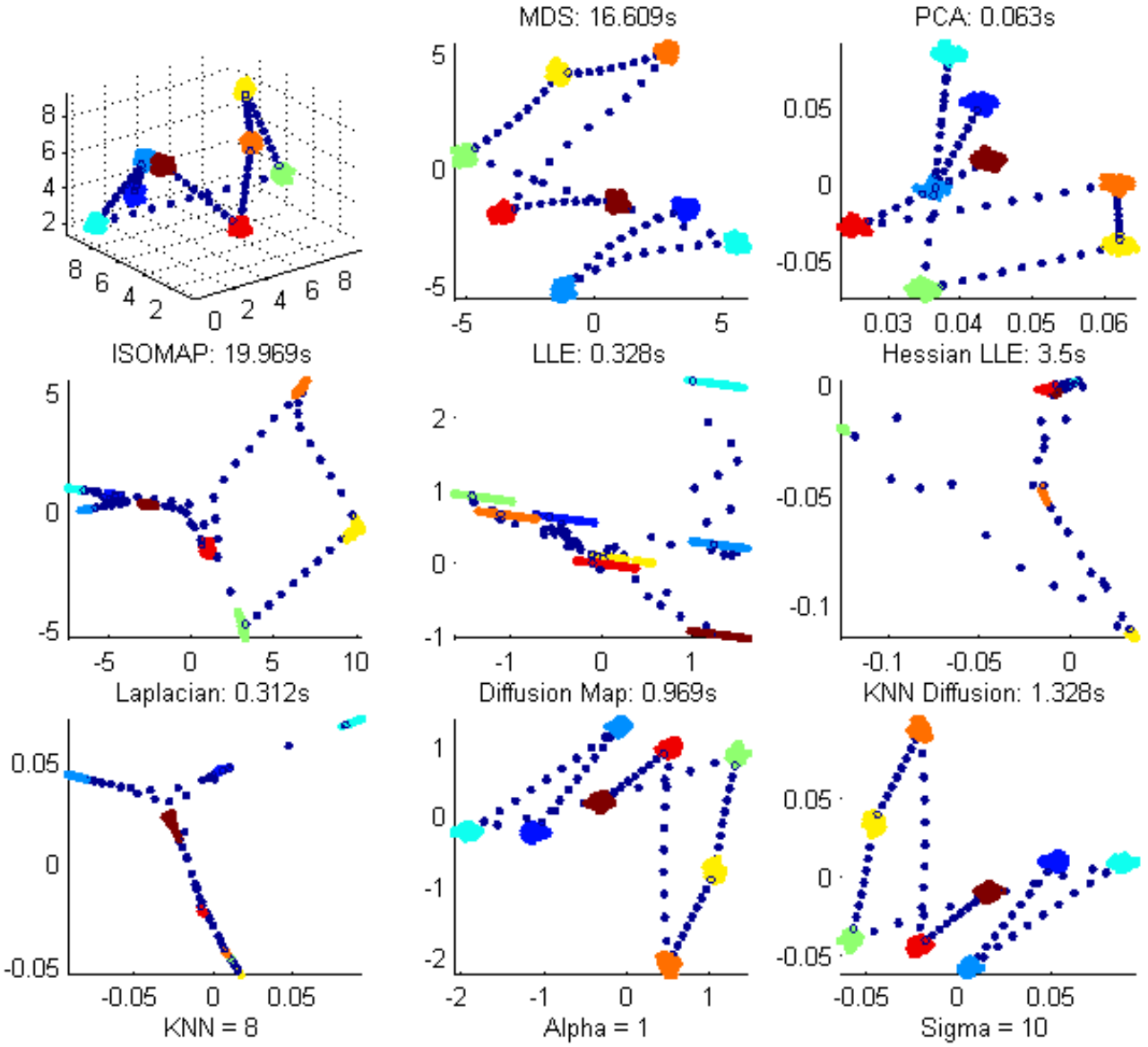
Clustering

A good mapping should preserve clusters in the original data set.
3D Clusters: Generate M non-overlapping clusters with random centers.
Connect the clusters with a line.





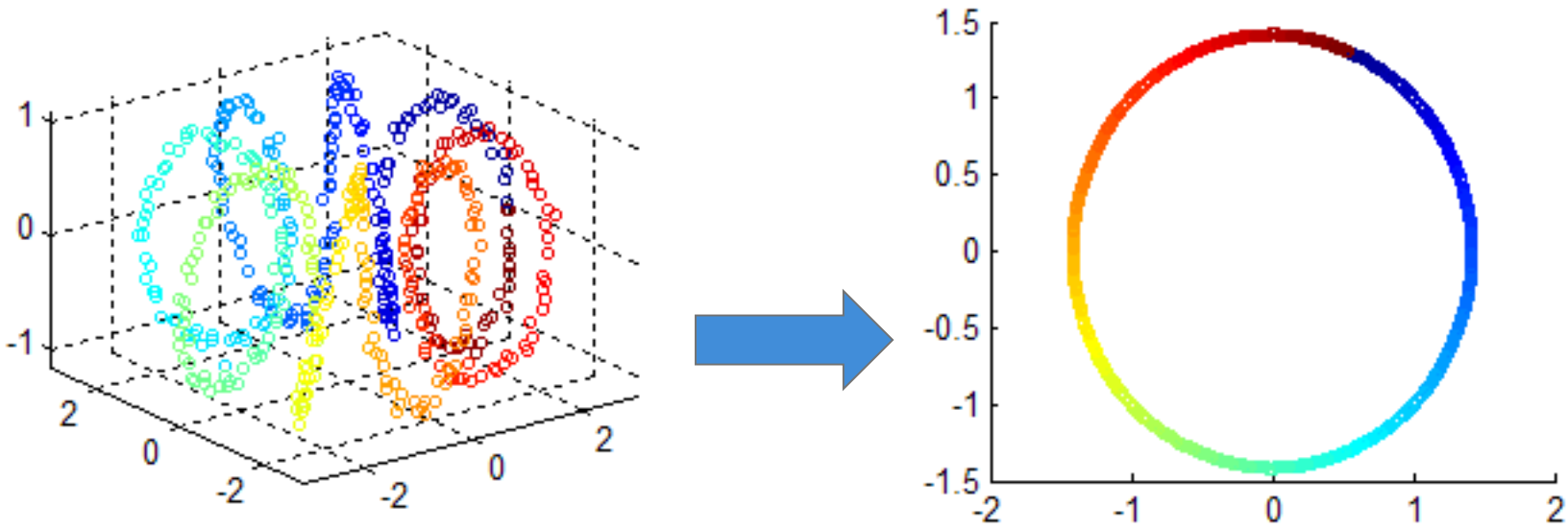
For M = 3 clusters, MDS and PCA can project correctly.
Diffusion Maps work well with large Sigma.
LLE compresses each cluster into a single point.
Hessian LLE has trouble with the sparse connecting lines.



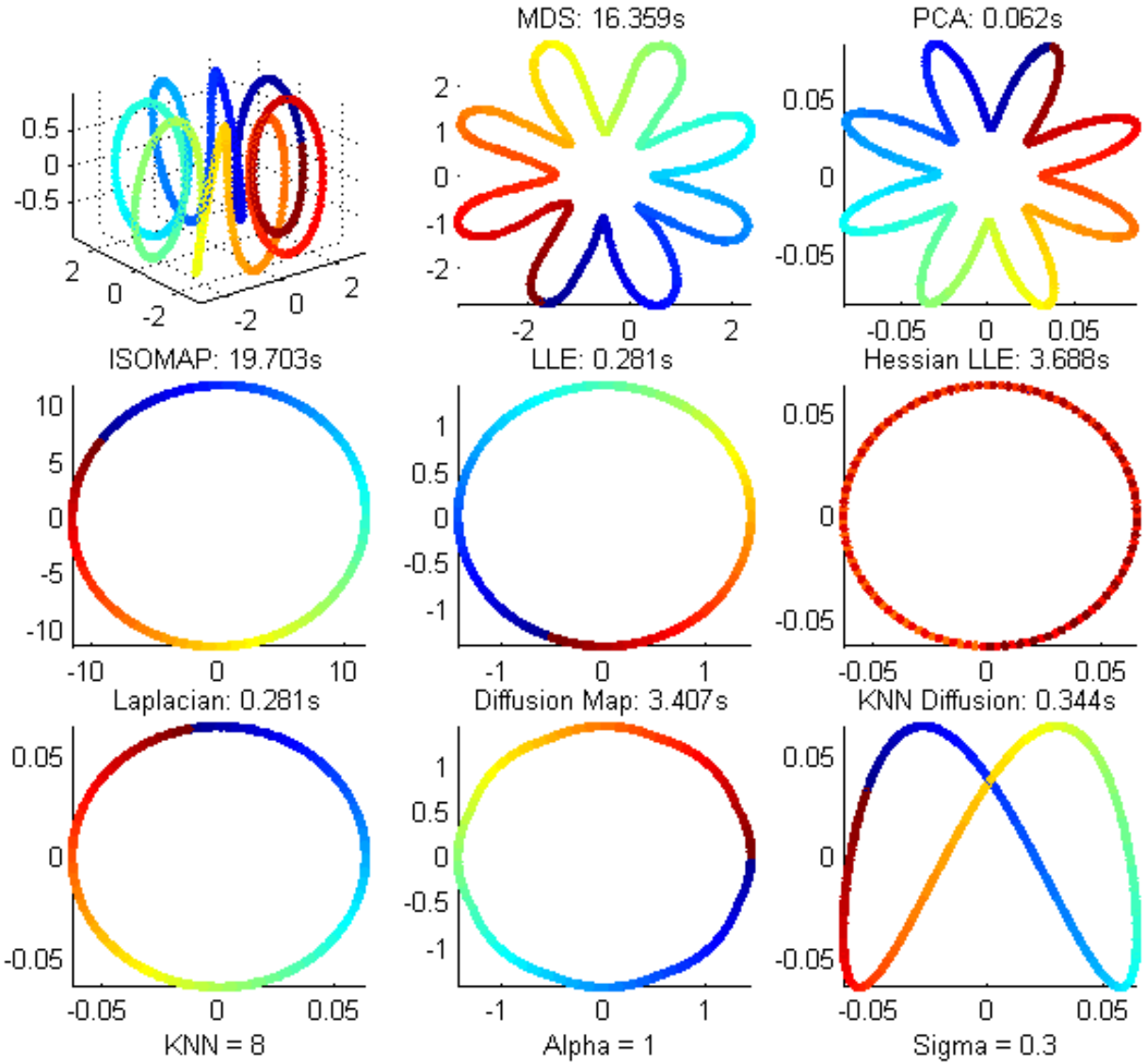
For M=8 clusters, MDS and PCA can still recover.
Diffusion Maps do quite well.
LLE and ISOMAP are decent, but Hessian and Laplacian fail.

Noise & Non-uniform Sampling

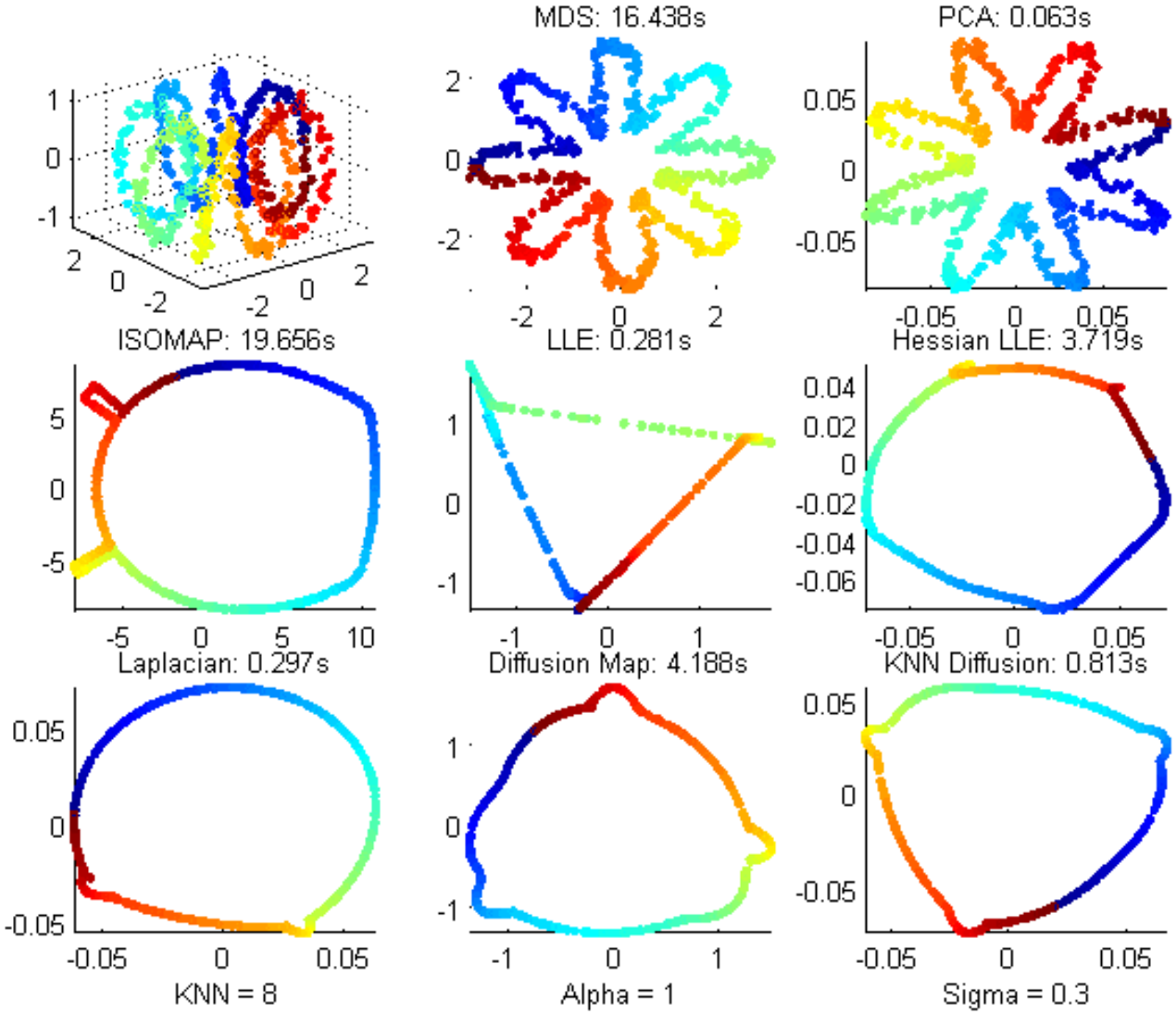
Can the method handle changes from dense to sparse regions?
Toroidal Helix should be unraveled into a circle parametrized by t.



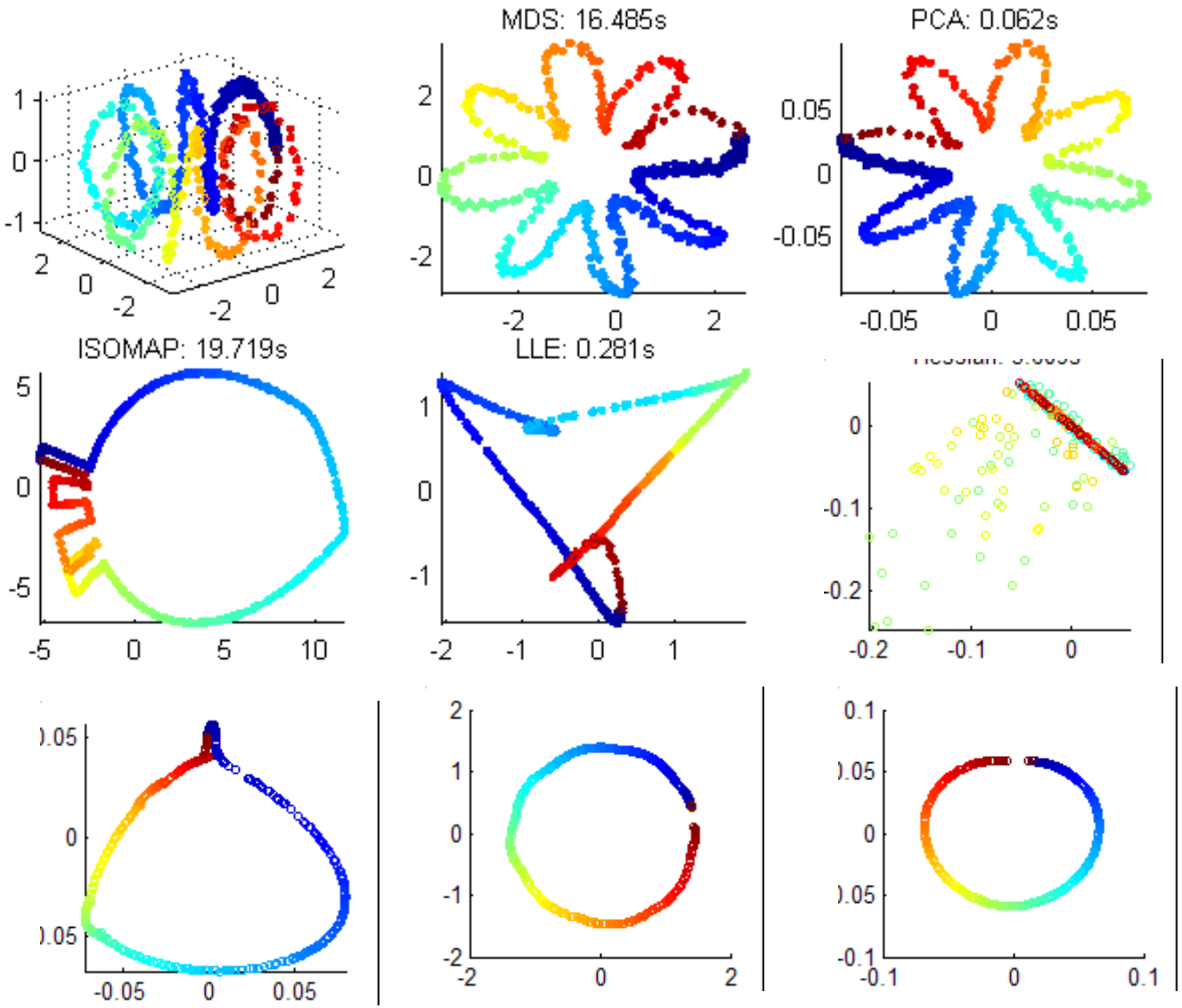
We can change the sampling rate along the helix by changing the exponent R on the parameter t and we can add some noise.



With no noise added, ISOMAP, LLE, Laplacian, and Diffusion Map are correct.
MDS and PCA project to an asterisk.
What's up with Hessian and KNN Diffusion?



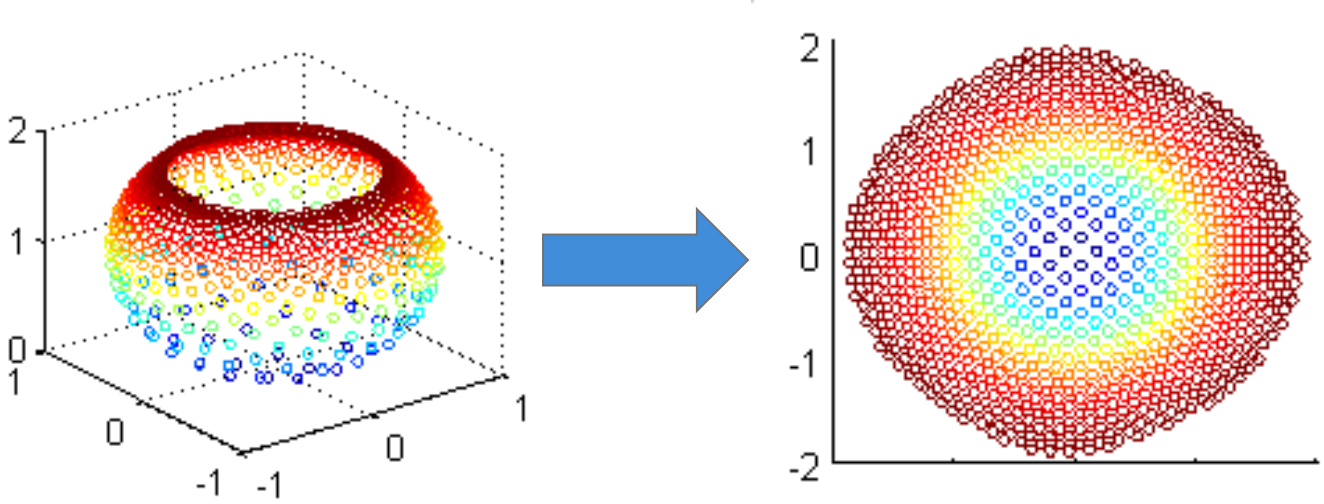
Adde noise to the Helix sampling.
LLE cannot recover the circle.
ISOMAP emphasizes outliers more than the other methods.

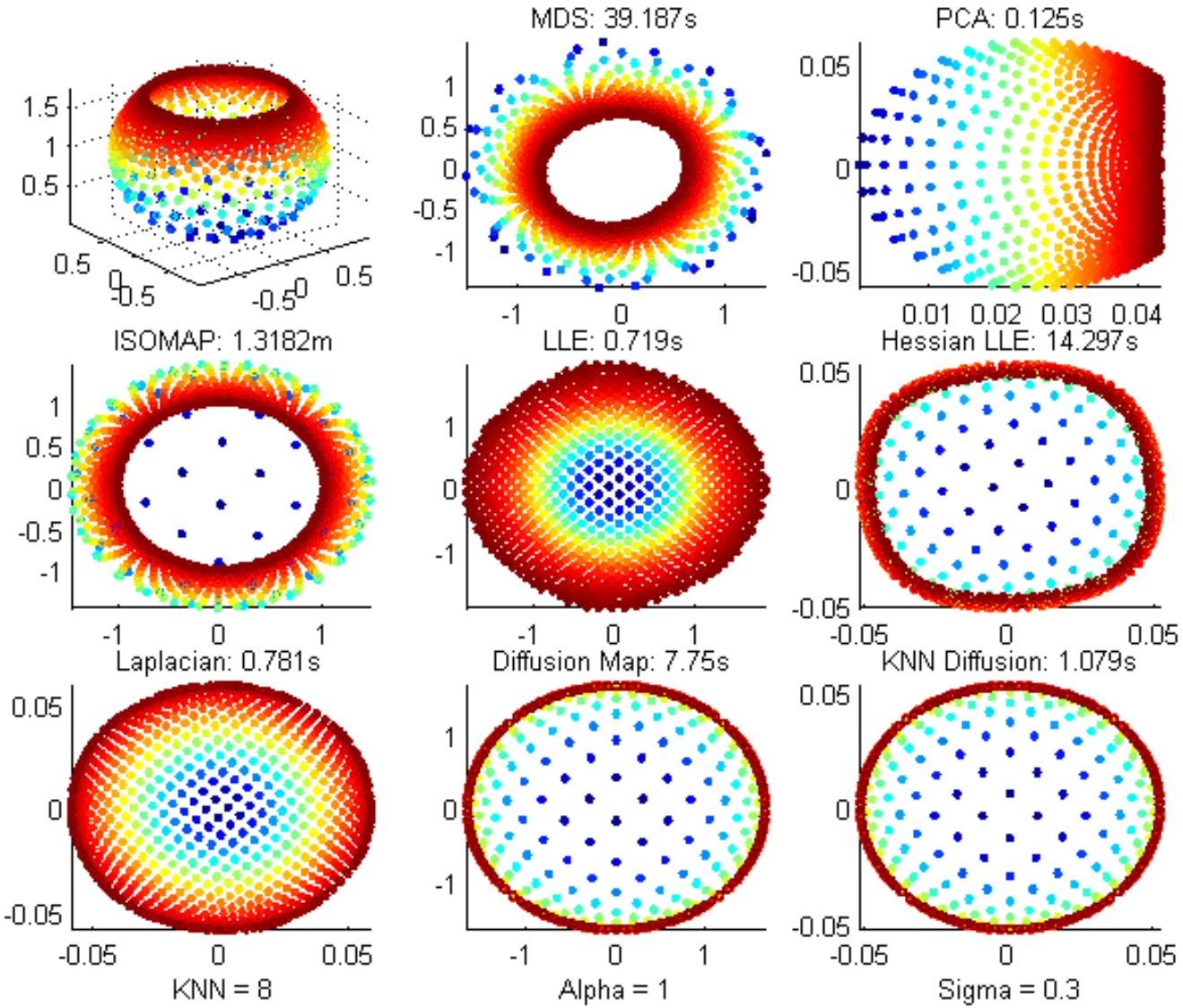


When the sampling rate is changed along the torus, Laplacian starts to mess up and Hessian is completely thrown off.
Hessian LLE code crashed frequently on this example.
Diffusion maps handle it quite well for carefully chosen Sigma=0.3.

Sparse Data & Non-uniform Sampling

Of course, we want as much data as possible. But can the method handle sparse regions in the data?
Punctured Sphere: the sampling is very sparse at the bottom and dense at the top.





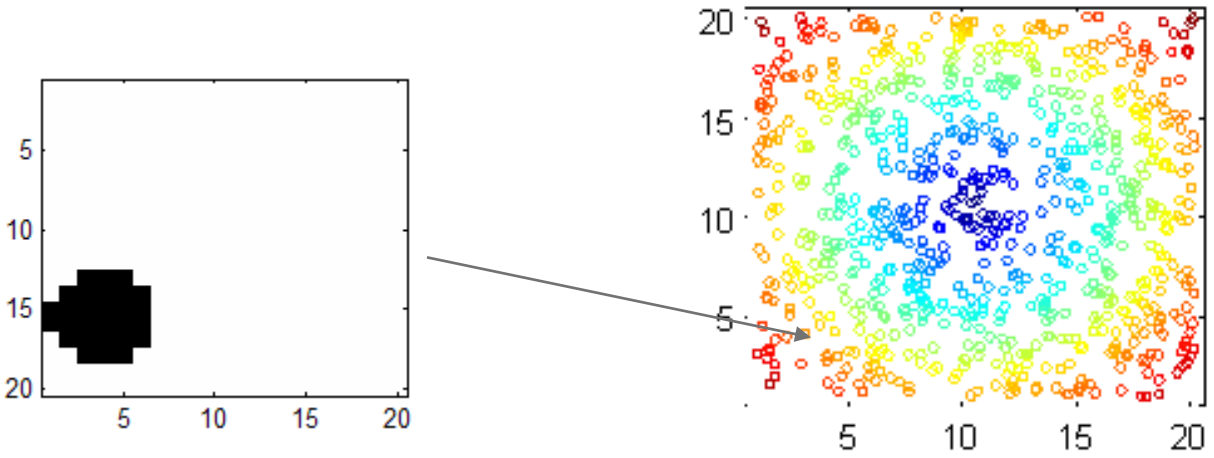
Only LLE and Laplacian get decent results.
PCA projects the sphere from the side. MDS turns it inside-out.
Hessian and Diffusion Maps get correct shape, but give too much emphasis to the sparse region at the bottom of the sphere.

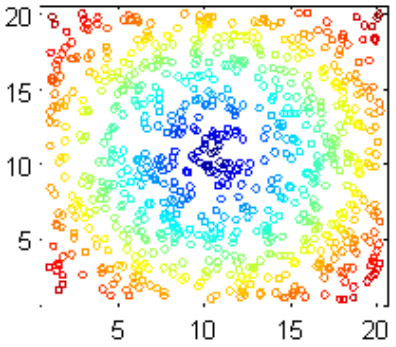
High-Dimensional Data

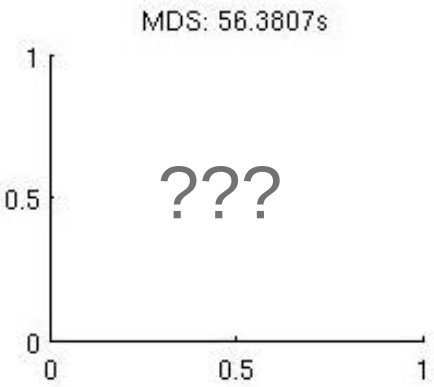
All of the examples so far have been 3D.
But can the data handle high-dimensional data sets, like images?

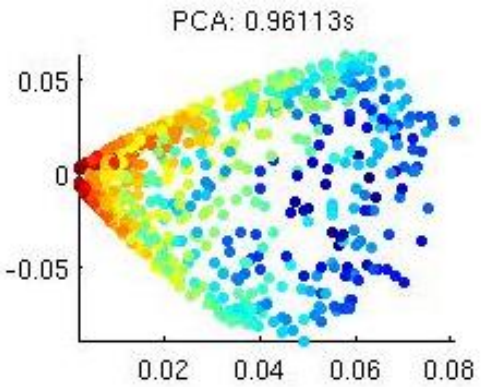
Disks: Create 20x20 images with a disk of fixed radius and random center.

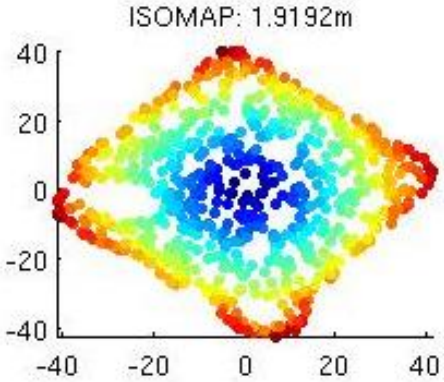
We should recover the centers of the circles.



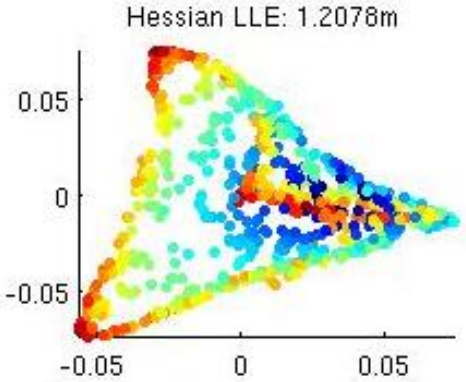


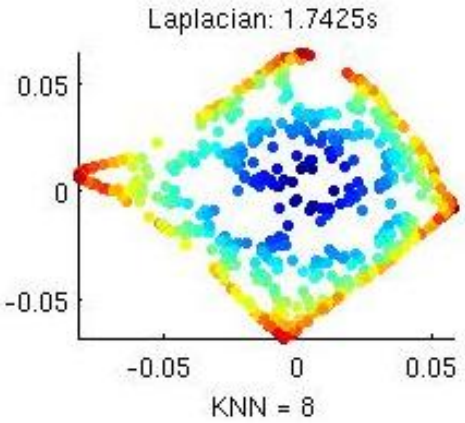


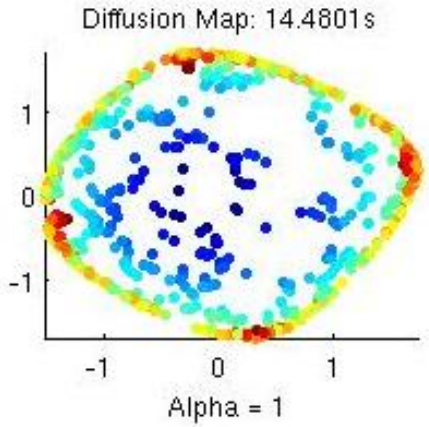




LLE Crashed



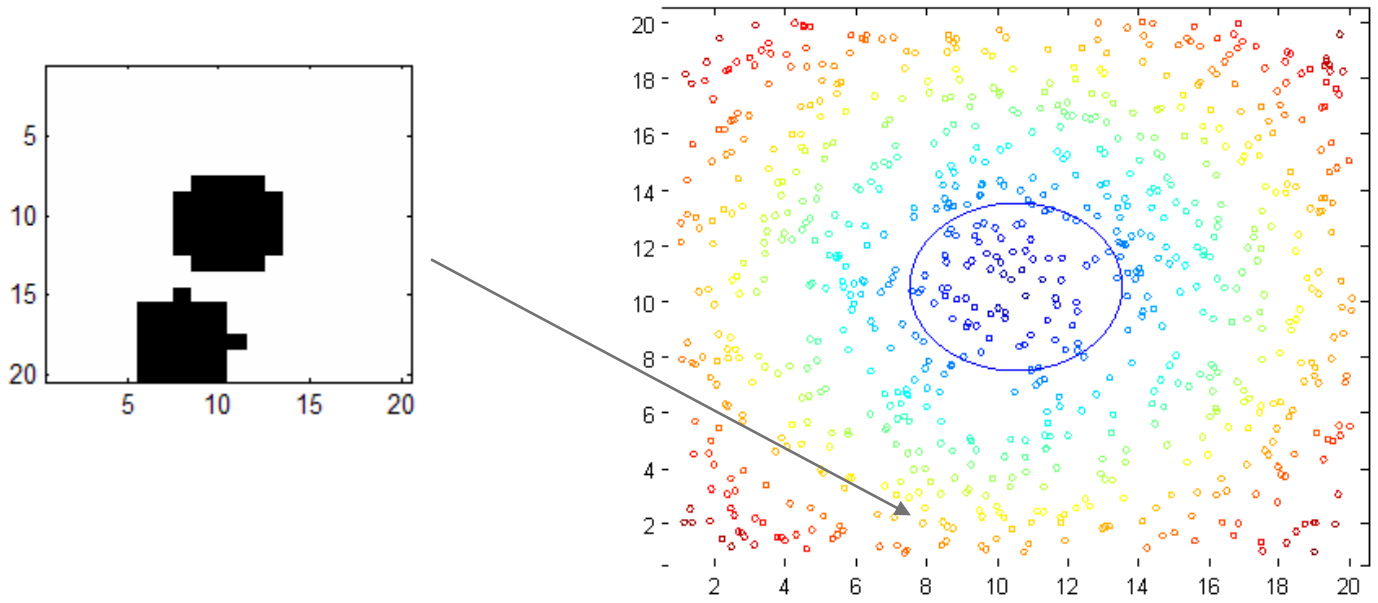


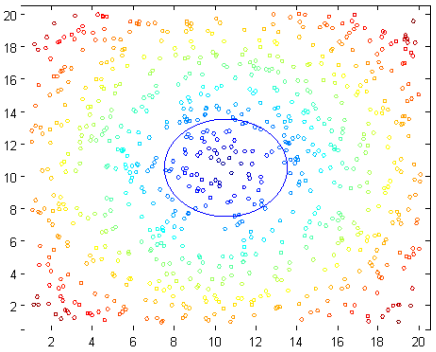


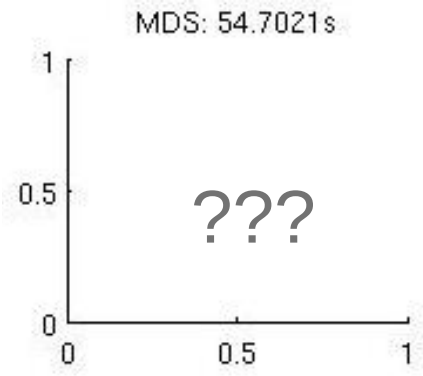
LLE crashed on high-dimensional data set.
Number of images was not high enough, but ISOMAP did a very good job.

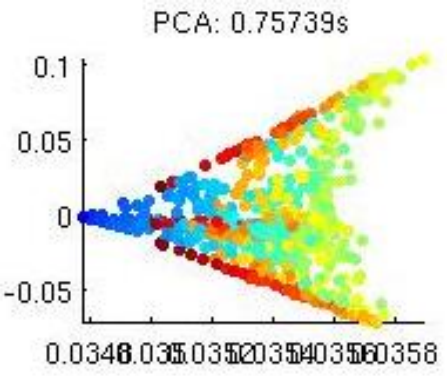
Occluded Disks

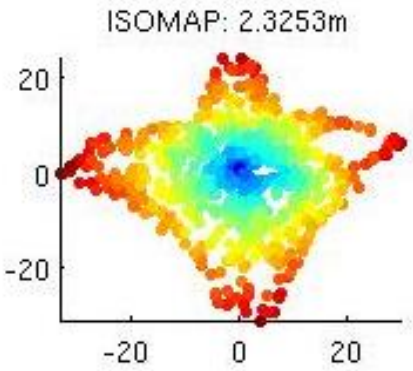
We can add a second disk of radius R in the center of every image.





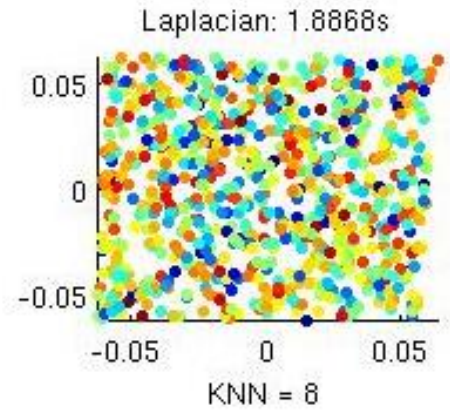


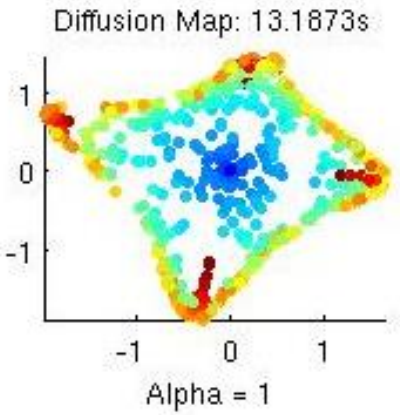




LLE Crashed

Hessian Crashed





Both LLE and Hessian crashed, possibly # points is too small.
Laplacian failed completely.
Is ISOMAP the best for high-dimensional data?

Sensitivity to Parameters

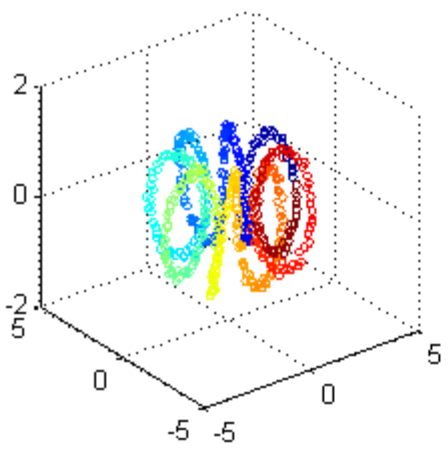
When the number of points is small or the data geometry is complex, it is important to set K appropriately, neither too big nor small.

But if the data set is dense enough, we expect K around 8 or 10 to suffice.

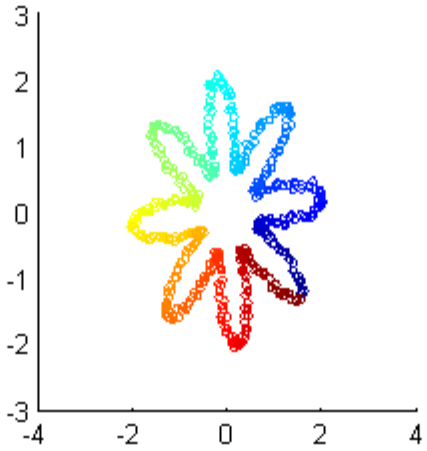
For Diffusion Maps, the method is very sensitive to the Sigma in Gaussian kernel. Varies from example to example.

Diffusion Map Sigma depends on manifold.

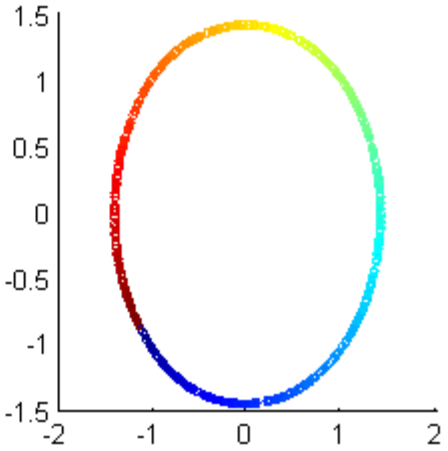
Helix



X

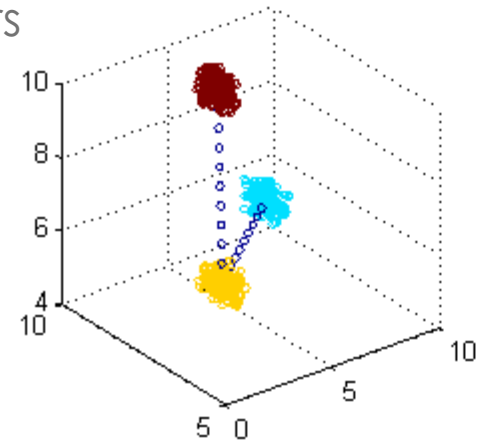


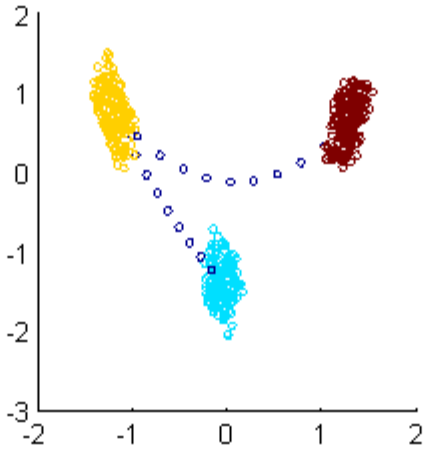
Sigma = 10

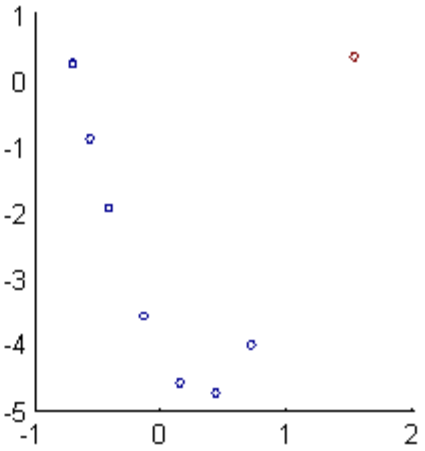


Sigma = 0.2

Clusters







So what have you learned, Dorothy?

	MDS	PCA	ISOMAP	LLE	Hessian	Laplacian	Diffusion Map	KNN Diffusion
Speed	Very slow	Extremely fast	Extremely slow	Fast	Slow	Fast	Fast	Fast
Infers geometry?	NO	NO	YES	YES	YES	YES	MAYBE	MAYBE
Handles non-convex?	NO	NO	NO	MAYBE	YES	MAYBE	MAYBE	MAYBE
Handles non-uniform sampling?	YES	YES	YES	YES	MAYBE	NO	YES	YES
Handles curvature?	NO	NO	YES	MAYBE	YES	YES	YES	YES
Handles corners?	NO	NO	YES	YES	NO	YES	YES	YES
Clusters?	YES	YES	YES	YES	NO	NO	YES	YES
Handles noise?	YES	YES	MAYBE	NO	YES	YES	YES	YES
Handles sparsity?	YES	YES	YES	YES	NO *may crash*	YES	NO	NO
Sensitive to parameters?	NO	NO	YES	YES	YES	YES	VERY	VERY

Some Notes on using MANI

- Hard to set K and Sigma just right.
- MDS and ISOMAP are very slow.
- Hessian LLE is pretty slow. Since Hessian needs a dense data set, this means it takes even longer when the # points is large.
- Occluded Disks is 400-dimensional data, which takes a long time and a lot of data points to correctly map.
- Matlab GUIs seem to run better on PC than Linux.

Credits

- M. Belkin,
- P. Niyogi,
- Todd Wittman
- Eunsu Kang

Thank you for your attention!