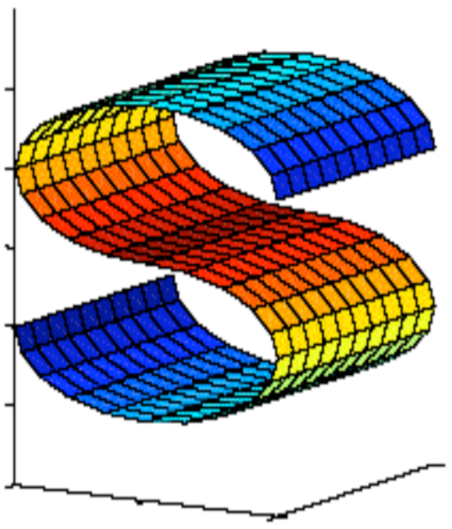
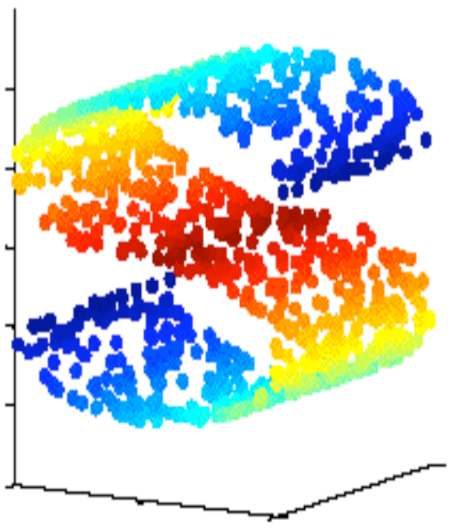
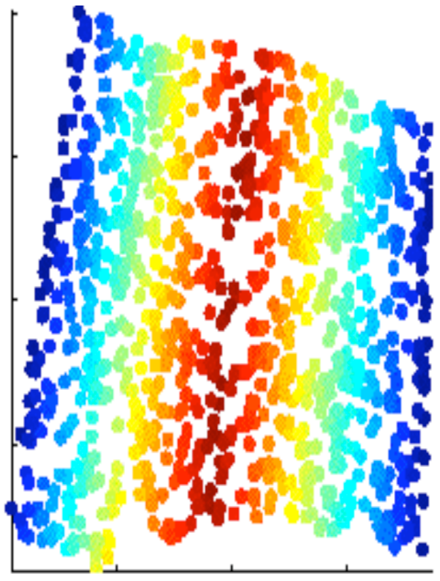
Unfolding a manifold by semidefinite programing

Prof. Lawrence Saul
Computer and Information Science
University of Pennsylvania

(with S. Roweis, K. Weinberger, F. Sha, and B. Packer)

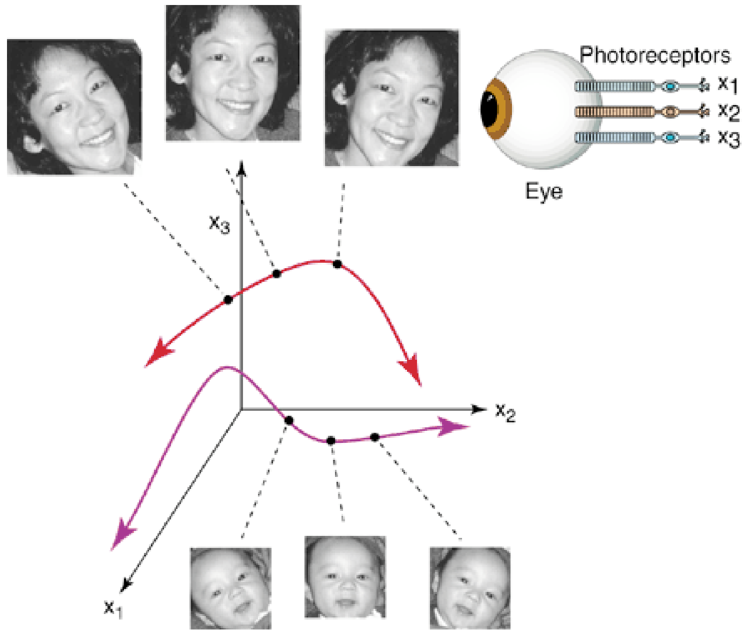
Statistics, Geometry, Computation

Given high dimensional data sampled from a low dimensional manifold, how to compute a faithful embedding?

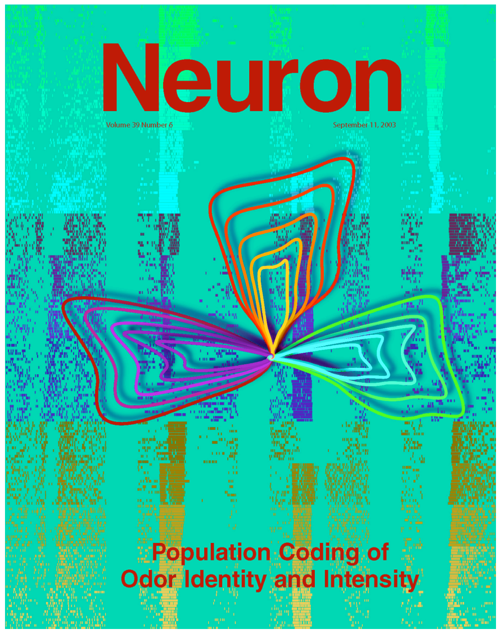




Applications

Low dimensional manifolds arise in many areas of information processing.



(Seung & Lee, 2000)



(Stopfer et al, 2003)

Dimensionality reduction

- **Inputs (high dimensional)**

  $$\vec{X}_i \in \mathbb{R}^D \text{ with } i = 1,2,\ldots,N$$

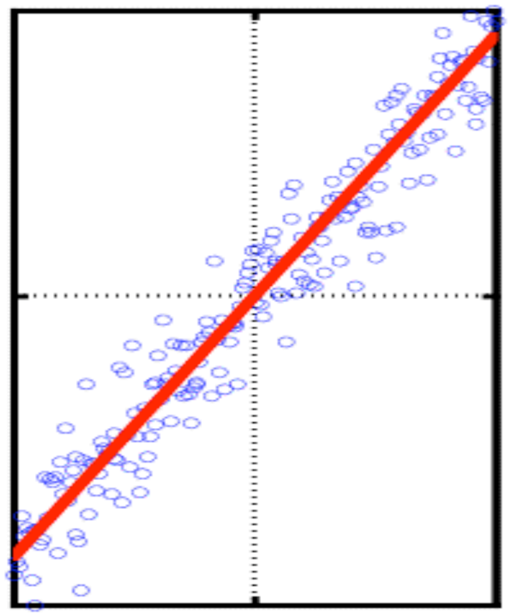
- **Outputs (low dimensional)**

  $$\vec{Y}_i \in \mathbb{R}^d \text{ where } d < D$$

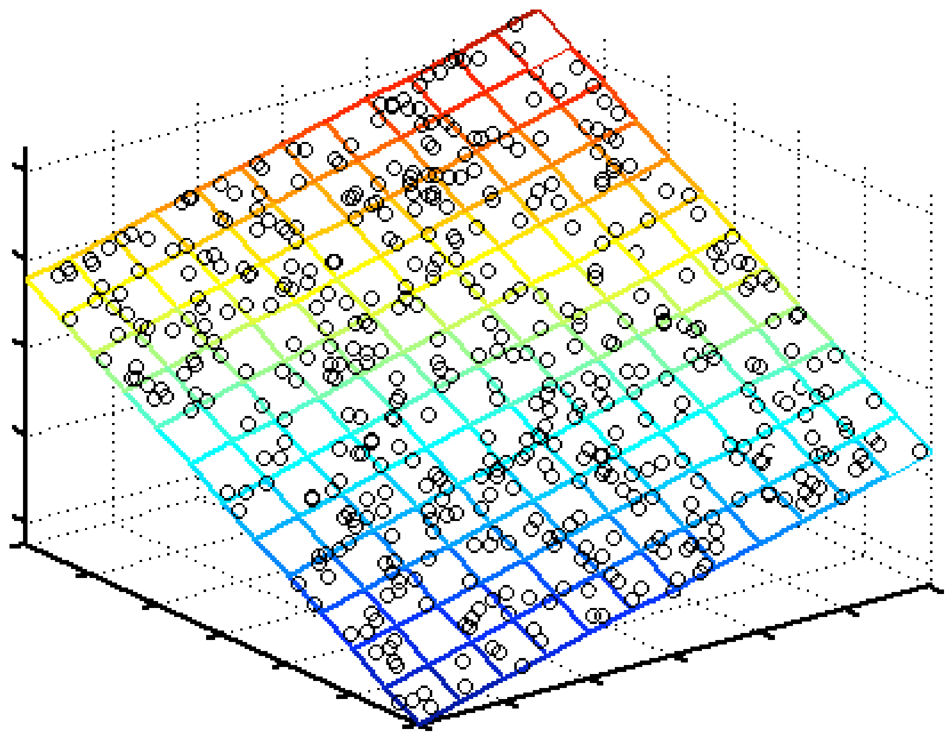
- **Embedding**

  Nearby points remain nearby.
  Distant points remain distant.
  (Estimate *d*.)

Subspaces



$D = 2$
$d = 1$



$D = 3$
$d = 2$

Linear methods

- **Principal component analysis**

  Project inputs into subspace of maximal variance:

$$\max\left(\mathrm{tr}\left[Y^T Y\right]\right) \text{ with } Y = PX$$

- **Multidimensional scaling**

  Project inputs into subspace that preserves pairwise distances:

$$\left|\vec{Y}_i \square \vec{Y}_j\right|^2 \square \left|\vec{X}_i \square \vec{X}_j\right|^2$$

Matrices of PCA and MDS

Correlation matrix: $C_{\alpha\beta} \sim \mathbf{E}[X_{\alpha} X_{\beta}]$

Gram matrix: $G_{ij} = \vec{X}_i \bullet \vec{X}_j$

These matrices have the same rank and nonzero eigenvalues.

Dimensionality reduction

- **Eigenvectors**

  eigs(C) = linear projections of PCA
  eigs(G) = projected outputs of MDS

- **Eigenvalues**

  Always nonnegative.
  Gaps indicate latent dimensionality.

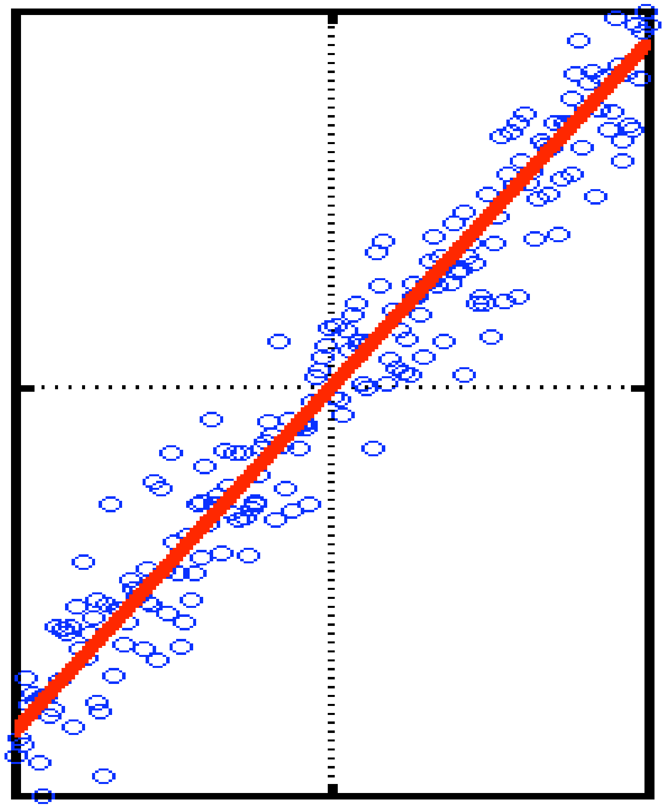
Different intuitions,
but equivalent results.

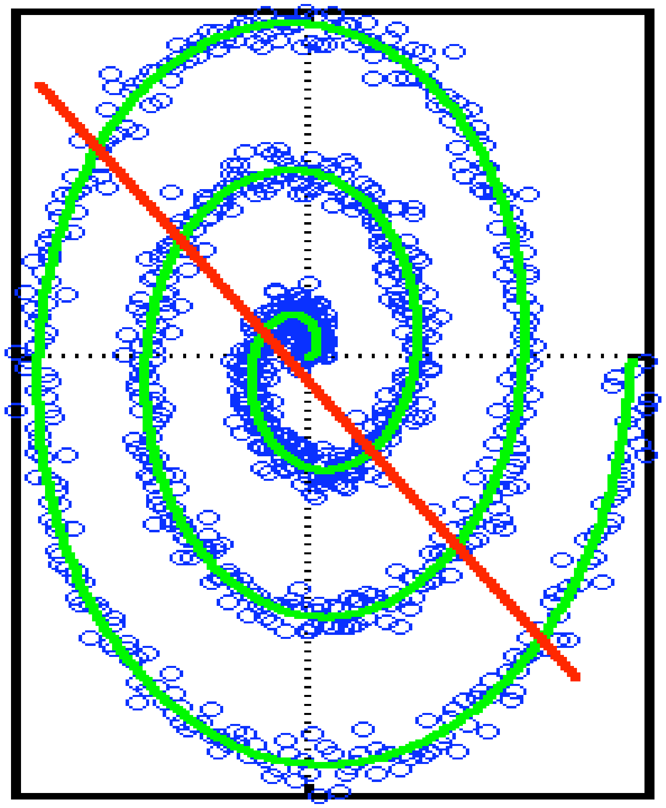
Properties of PCA and MDS

- Strengths
  - Eigenvector methods
  - Non-iterative
  - No local optima
  - No “free” parameters
- Weakness

  PCA and MDS are linear methods.

Subspaces vs Manifolds





Linear methods are limited.

Questions

- Are there eigenvector methods for nonlinear dimensionality reduction?

$(\text{Yes})^n$ with $n \geq 8$

- Equally simple as PCA and MDS?

Almost!

Recent Algorithms

- In this talk

  Locally linear embedding (LLE)

  Semidefinite embedding (SDE)

- Related work by others

  Isomap (Tenebaum , de Silva, & Langford)

  Laplacian eigenmaps (Belkin & Niyogi)

  Local tangent space alignment (Zhang & Zha)

  Hessian LLE (Donoho & Grimes)

  Charting (Brand)

Outline of talk

- **Thesis**

  LLE preserves local linearity relations. Constructs, diagonalizes a sparse matrix.

- **Antithesis**

  SDE preserves local distances, angles. Constructs, diagonalizes a dense matrix.

- **Synthesis**

  Exploit symmetries of LLE to speed up SDE by several orders of magnitude.

Algorithm #1: LLE

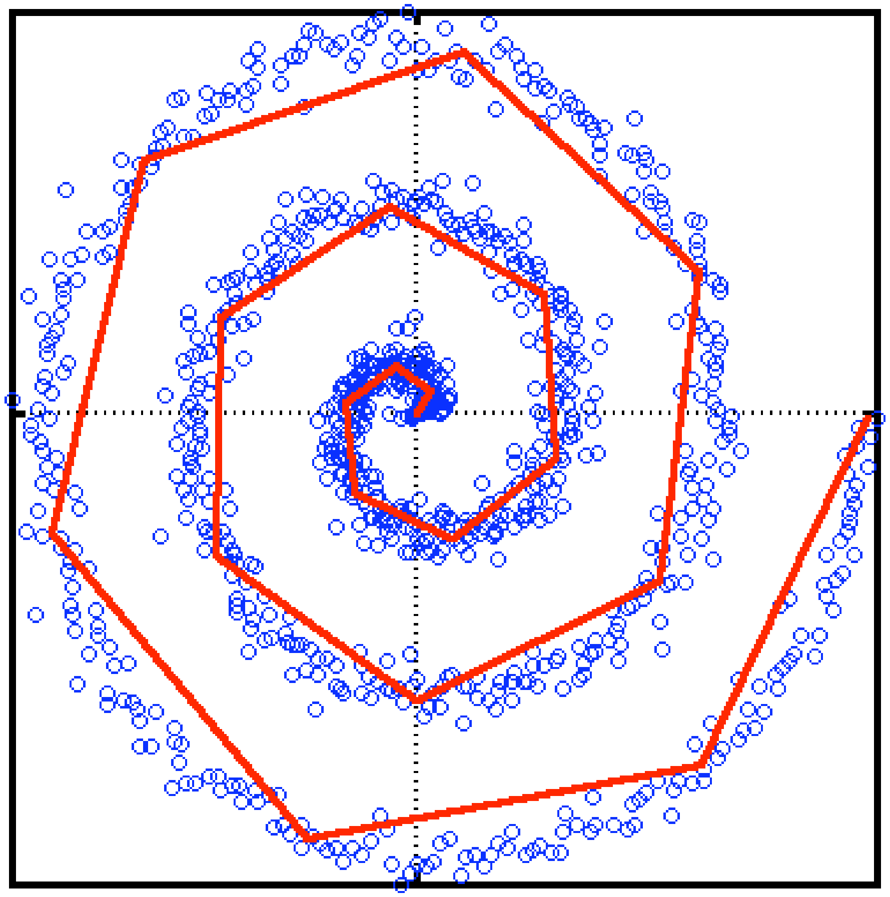
Locally Linear Embedding

“Think globally, fit locally.”

Local linearity

A manifold is locally linear, even if globally nonlinear.

How can we use this?



Locally Linear Embedding (LLE)

- Steps
  1. Nearest neighbor search.
  2. Least squares fits.
  3. Sparse eigenvalue problem.
- Properties
  - Obtains highly nonlinear embeddings.
  - Non-iterative, not prone to local minima.

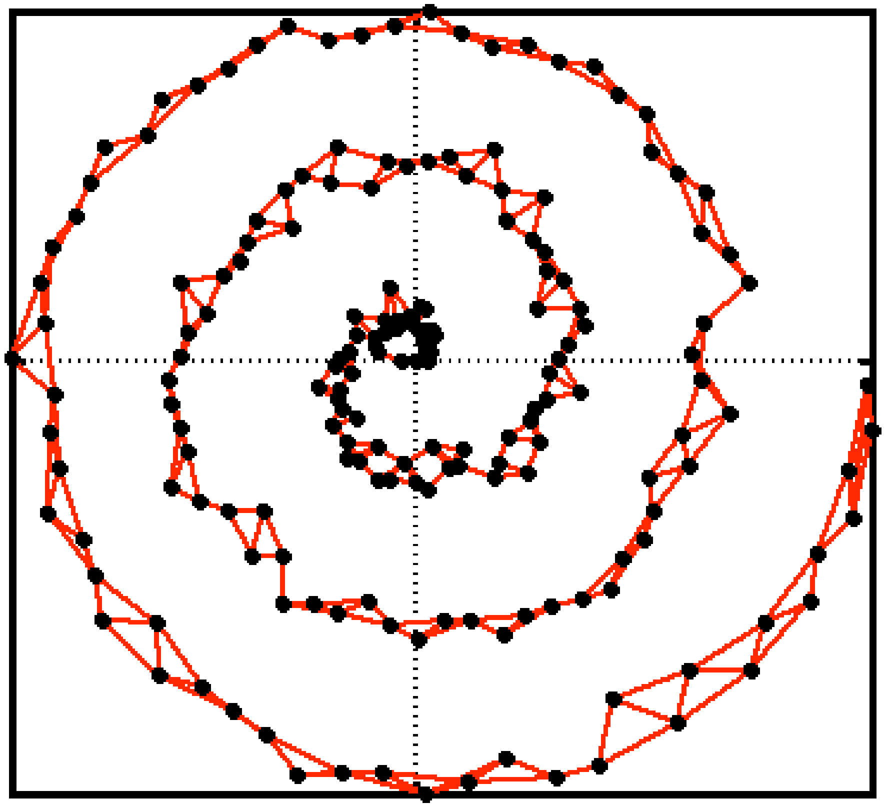
Step 1. Identify neighbors.

- Examples of neighborhoods
  - *K* nearest neighbors
  - Neighbors within radius *r*
  - Metric based on prior knowledge
- Assumptions
  - Data is sampled from a manifold.
  - Manifold is well sampled.

Nearest neighbor graph

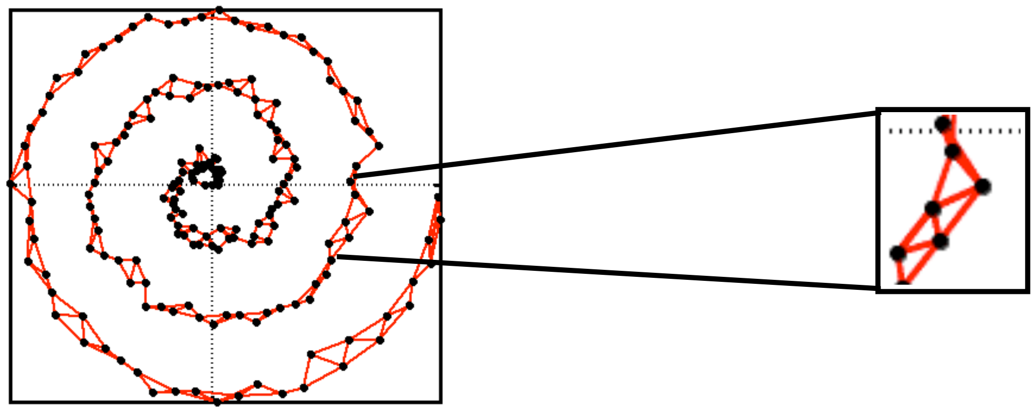
Assumptions:

- **Graph is connected.**
- **Neighborhoods on the graph correspond to neighborhoods on the manifold.**



Step 2. Compute weights.

- Characterize local geometry of each neighborhood by weights W_{ij}.



- Compute weights by reconstructing each input (linearly) from neighbors.

Linear reconstructions

- Local linearity

  Neighbors lie on locally linear patches of a low dimensional manifold.

- Reconstruction errors

  Least squared errors should be small:

$$\varepsilon(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2$$

Least squares fits

- **Choose weights to minimize errors:**

$$\varepsilon(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2$$

- **Constraints:**

Nonzero W$_{ij}$ only for neighbors.
Weights must sum to one: $\sum_j W_{ij} = 1$

Symmetry

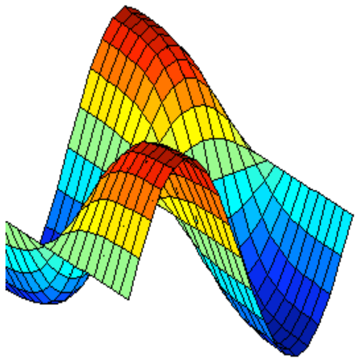
- **Cost per input**

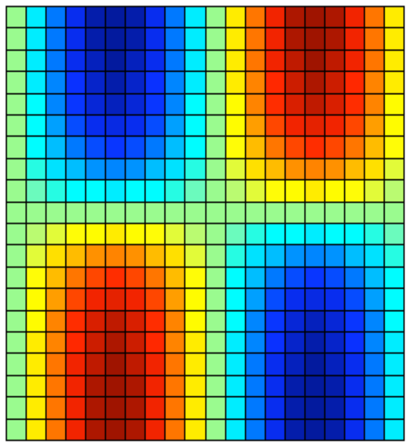
$$\varepsilon_i(W) = \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2$$

- **Local invariance**

Optimal weights W_{ij} are invariant to rotations, translations, and dilations.

Manifolds





- **Local linearity**

  Each neighborhood map looks like a translation, rotation, and dilation.

- **Local geometry**

  These transformations do not affect the weights W_{ij}: they remain valid.

Step 3. Compute the embedding.

- **Embedding**
  Map inputs to outputs: $\vec{X}_i \in \mathbb{R}^D$ to $\vec{Y}_i \in \mathbb{R}^d$

- **Minimize reconstruction errors.**
  Optimize outputs Y_i for fixed weights W_{ij}:

$$\Phi(Y) = \sum_i \left| \vec{Y}_i - \sum_j W_{ij} \vec{Y}_j \right|^2$$

- **Constraints**

  Center outputs on origin: $\sum \vec{Y}_i = \vec{0}.$
  Impose unit covariance matrix: $\frac{1}{N} \sum_i \vec{Y}_i \vec{Y}_i^T = I_d.$

Sparse eigenvalue problem

- **Quadratic form**

$$\Phi(Y) = \sum_{ij} M_{ij}\left(\vec{Y}_i \bullet \vec{Y}_j\right) \text{ with } M = (I - W)^T (I - W)$$

- **Rayleigh-Ritz theorem**

Optimal embedding given by bottom d+1 eigenvectors.

- **Solution**

**Discard bottom eigenvector [1 1 … 1].
Other eigenvectors satisfy constraints.**

Summary of LLE

- Three steps
  1. Compute K nearest neighbors.
  2. Compute weights W_{ij}.
  3. Compute outputs Y_i.
- Optimizations

$$\boxed{\varepsilon(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2}$$

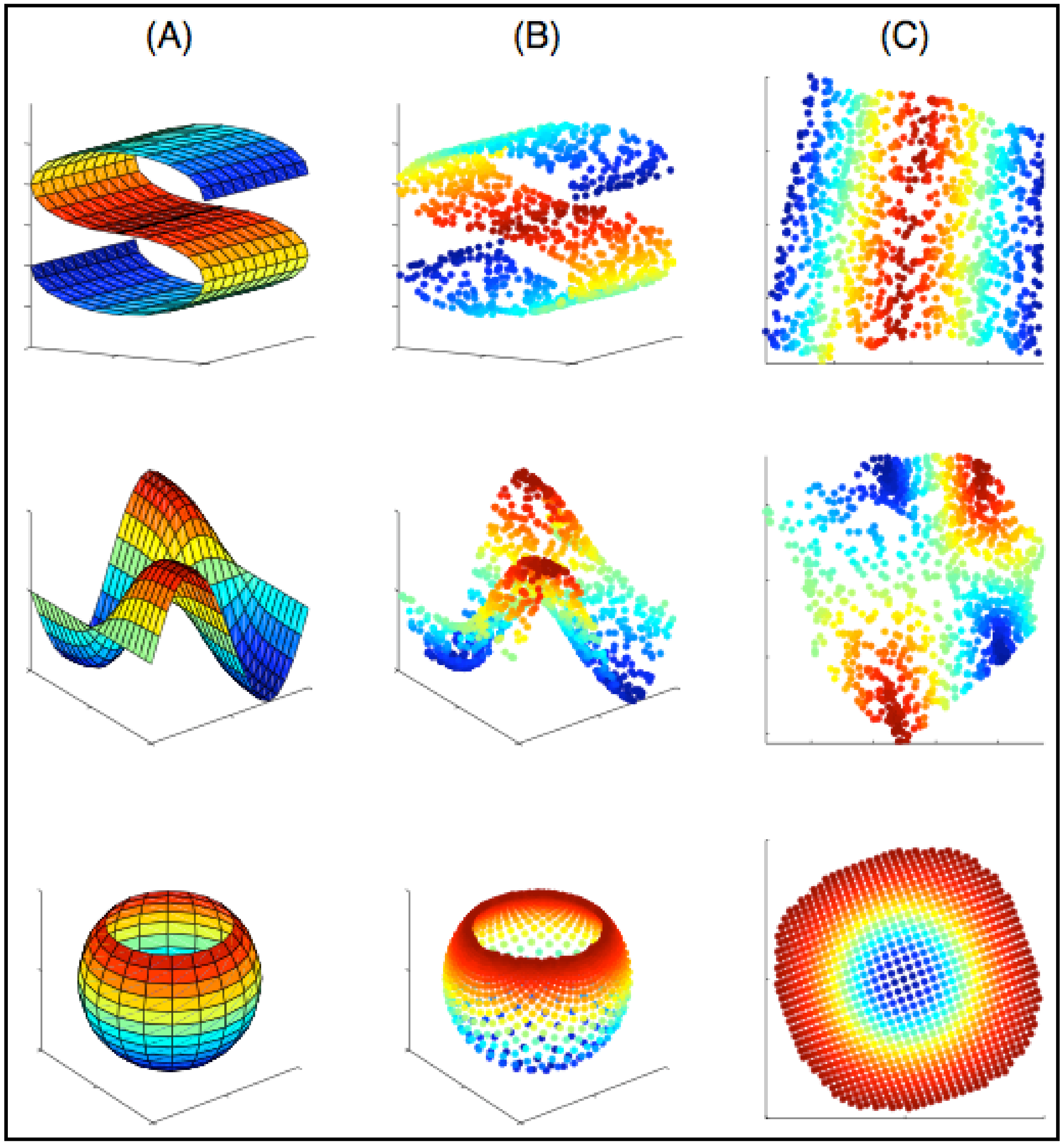
$$\boxed{\Phi(Y) = \sum_i \left| \vec{Y}_i - \sum_j W_{ij} \vec{Y}_j \right|^2}$$

Surfaces

N=1000
inputs

K=8
neighbors

D=3
d=2



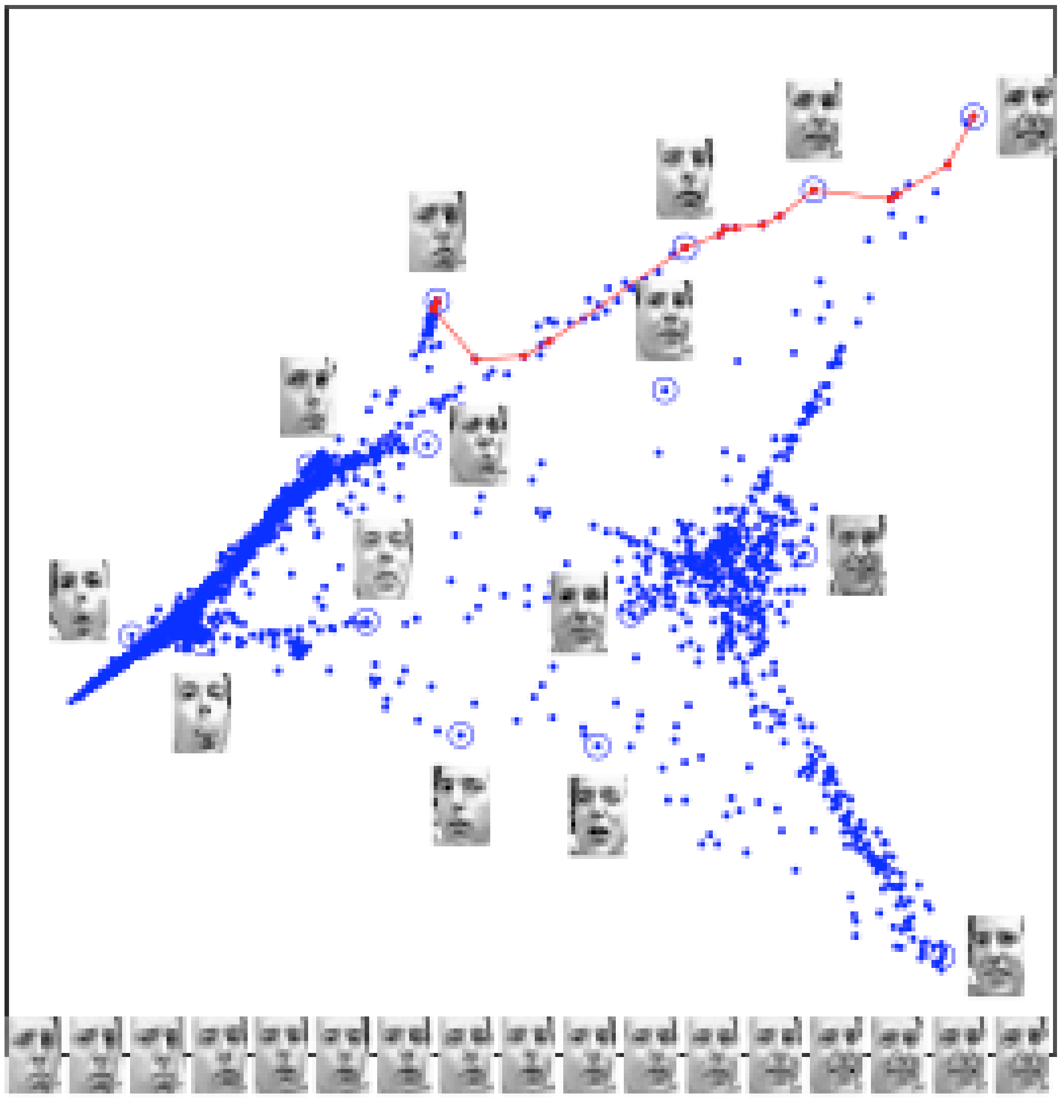
Pose and expression

N=1965
images

K=12
neighbors

D=560
pixels

d=2
(shown)



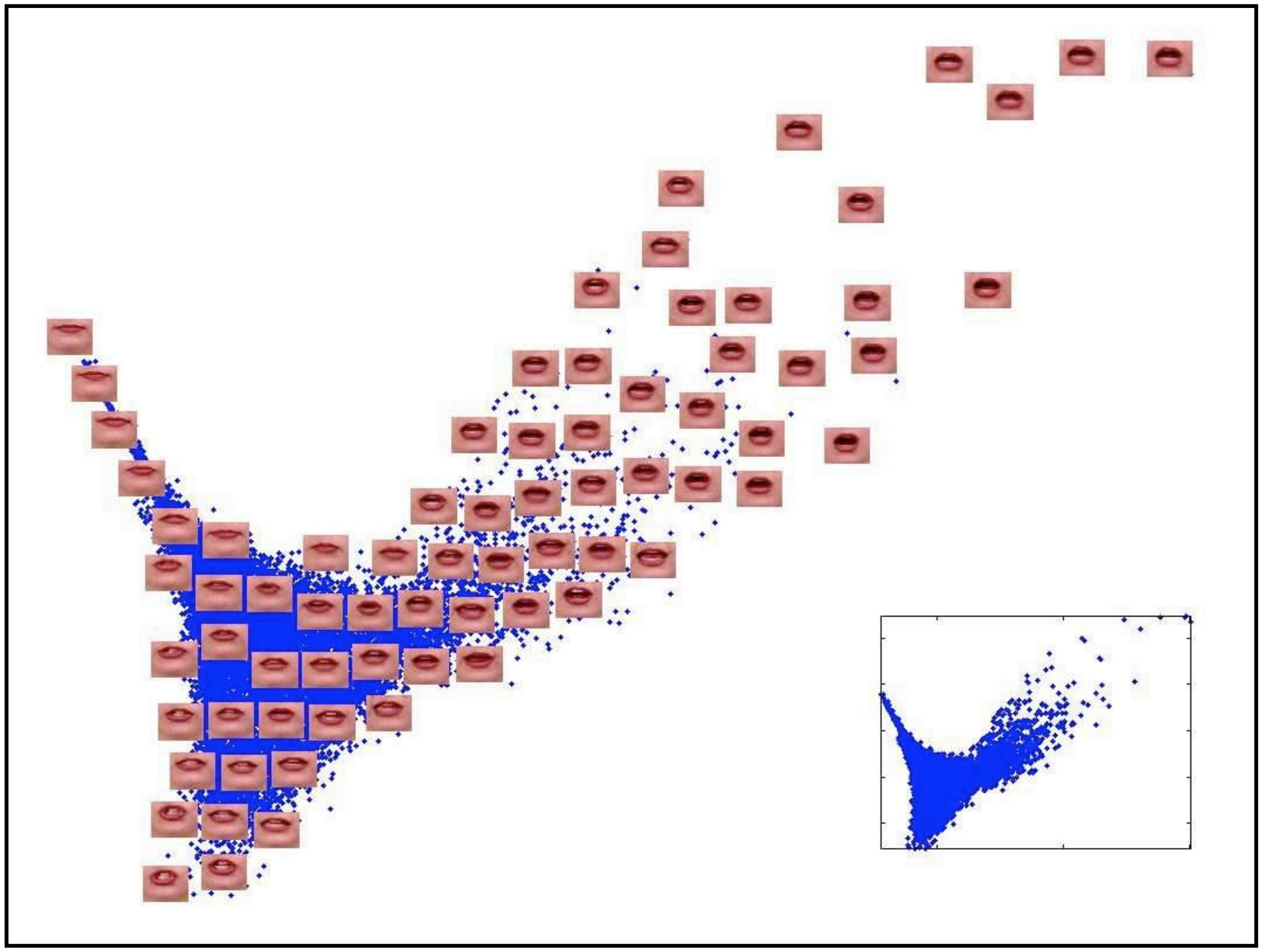
Lips

N=15960
images

K=24
neighbors

D=65664
pixels

d=2
(shown)



Summary of LLE

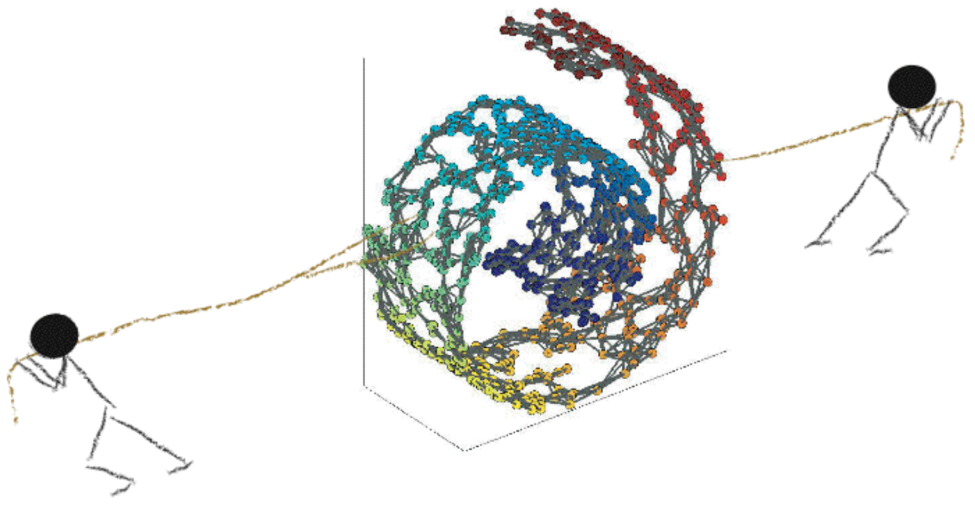
- Three steps:
  1. k-nearest neighbors of inputs X_i.
  2. Least squares fits for weights W_{ij}.
  3. Sparse eigensystem for outputs Y_i.
- Local symmetries:
  - translation
  - rotation
  - dilation

"Think globally, fit locally."

Algorithm #2: SDE

Semidefinite Embedding

"Maximum variance unfolding."



Motivation

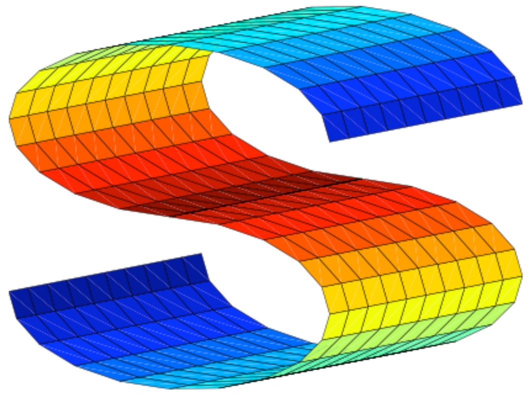
What class of mappings:

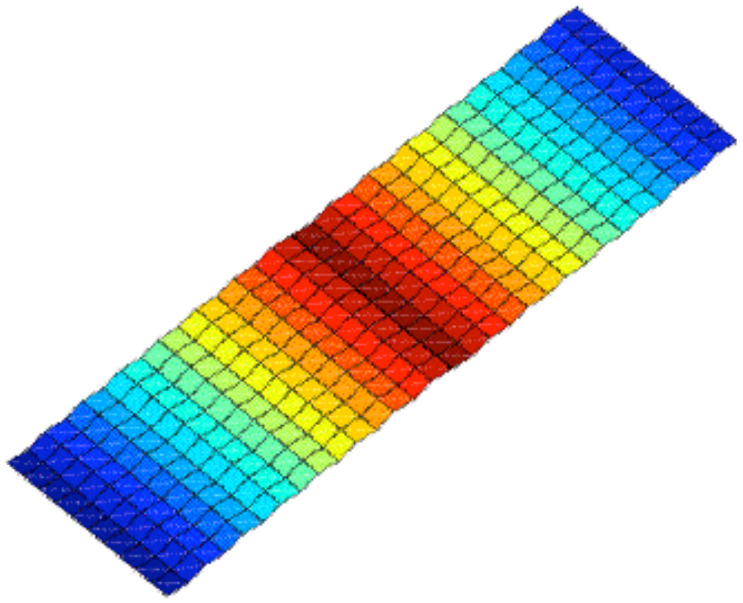
- Includes rotations and translations as a special case?
- Unravels manifolds into subsets of Euclidean space?

Isometry

- Intuitively

  Whatever you can do to a sheet of paper without holes, tears, or self-intersections.





Isometry (con’t)

- Informally

  A smooth, invertible mapping that preserves distances and looks *locally* like a rotation plus translation.

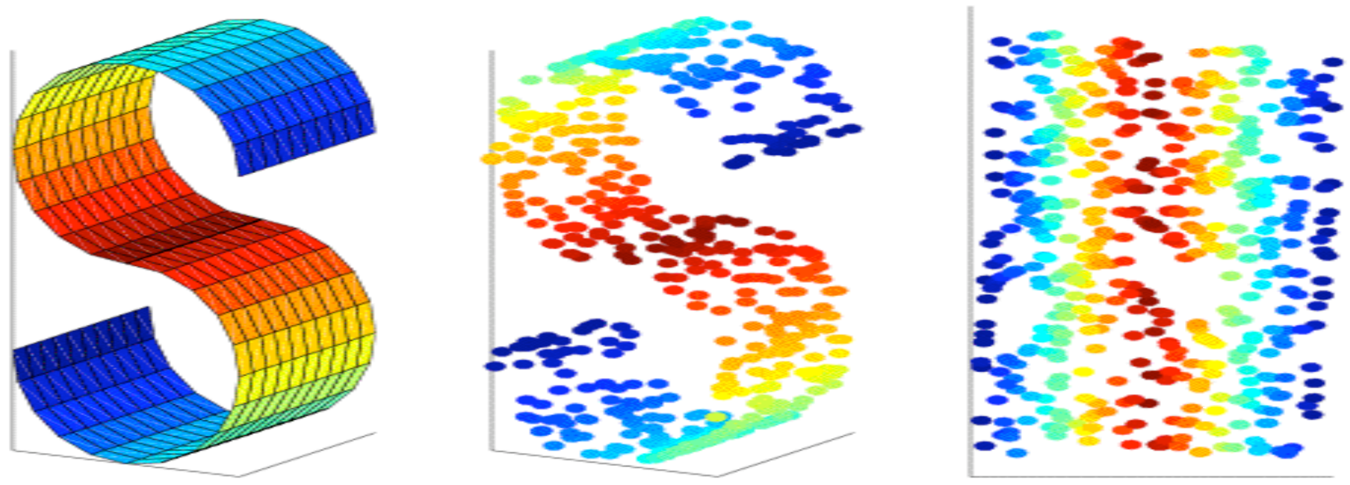
- Formally

  Two Riemannian manifolds are isometric if there is a diffeomorphism that pulls back the metric on one to the other.

Data on manifolds

From the continuous to the discrete:

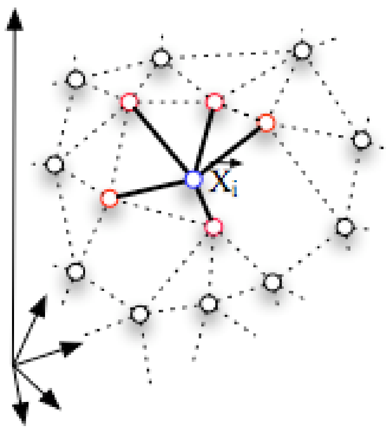
Isometry is defined between manifolds.
Can we extend the relation to data sets?



Discretely sampled manifolds

- **Neighborhood graph**

  Connect each point to its k nearest neighbors.



- **Locally isometric**

  Consider an embedding *Y* of *X* locally isometric if:

$$\boxed{\left(\vec{Y}_i \square \vec{Y}_j\right) \bullet \left(\vec{Y}_i \square \vec{Y}_k\right) = \left(\vec{X}_i \square \vec{X}_j\right) \bullet \left(\vec{X}_i \square \vec{X}_k\right)}$$

for all $\vec{X}_i$ with neighbors $\vec{X}_j$ and $\vec{X}_k$.

Dot product constraints

- **Gram matrices**

$$G_{ij} = \vec{X}_i \bullet \vec{X}_j \quad \text{(inputs)}$$

$$K_{ij} = \vec{Y}_i \bullet \vec{Y}_j \quad \text{(outputs)}$$

- **Locally isometric**

Consider an embedding *Y* of *X* locally isometric if:

$$\boxed{K_{ii} \square K_{ij} \square K_{ik} + K_{jk} = G_{ii} \square G_{ij} \square G_{ik} + G_{jk}}$$

for all $\vec{X}_i$ with neighbors $\vec{X}_j$ and $\vec{X}_k$.

Manifold learning

- Input

  Vectors $\vec{X}_i$ and Gram matrix $G_{ij} = \vec{X}_i \bullet \vec{X}_j$; latter determines former up to rotation.

- Problem

  Given $G_{ij} = \vec{X}_i \bullet \vec{X}_j$, how to construct $K_{ij} = \vec{Y}_i \bullet \vec{Y}_j$ such that Y "unfolds" the manifold of X?

- Algorithm

  What to optimize?
  What to constrain?

Constraints on K_{ij}

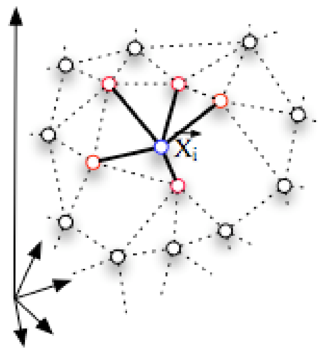
- Centered

Constrain outputs to have zero mean:

$$\sum_i \vec{Y}_i = \vec{0} \ \text{ implies } \ \left| \sum_i \vec{Y}_i \right|^2 = \sum_{ij} \vec{Y}_i \bullet \vec{Y}_j = \boxed{\sum_{ij} K_{ij} = 0}$$

- Locally isometric

Preserve local angles and distances:

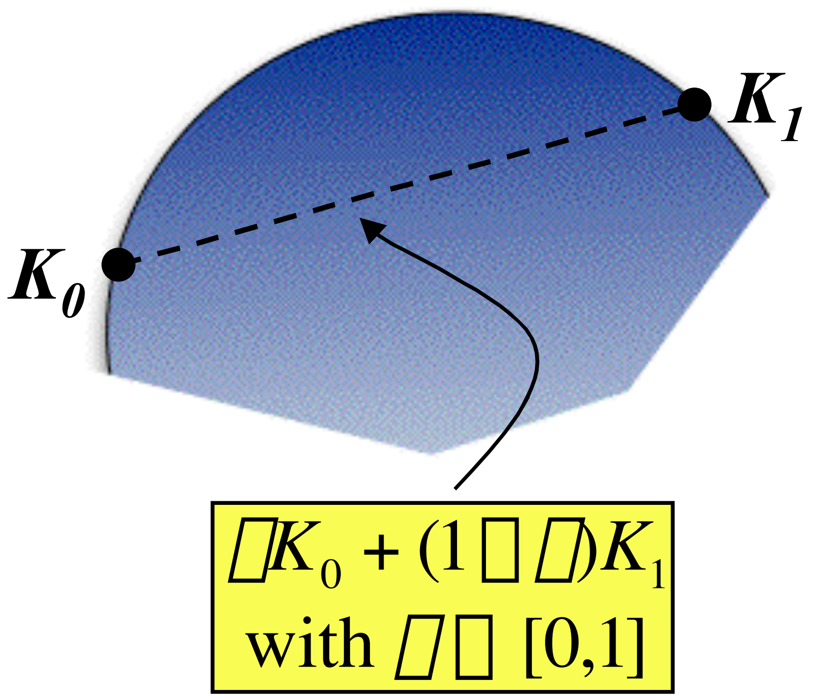


$$\boxed{K_{ii} - K_{ij} - K_{ik} + K_{jk} = G_{ii} - G_{ij} - G_{ik} + G_{jk}}$$

Constraints (con’t)

- Semidefinite

Eigenvalues of K must be nonnegative.



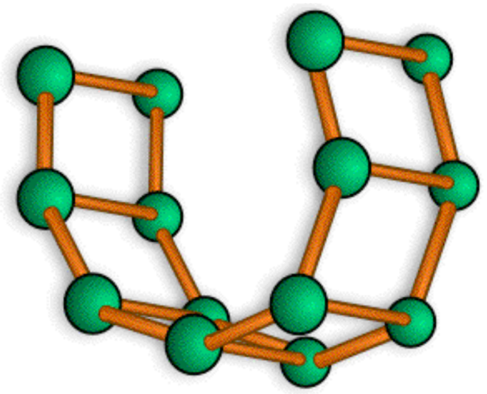
Semidefinite and linear constraints are convex.

O(Nk²) **constraints**

O(N²) **variables**

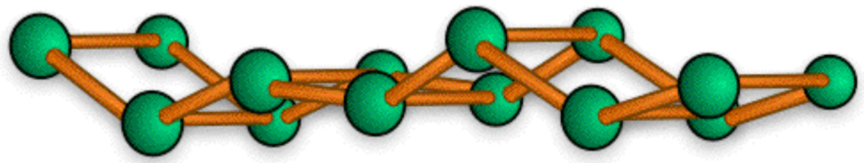
Unfolding a manifold

What function of the Gram matrix is being optimized below?



Before

$$G_{ij} = \vec{X}_i \bullet \vec{X}_j$$



After

$$K_{ij} = \vec{Y}_i \bullet \vec{Y}_j$$

Optimization

- Pull points apart

  Maximize sum of pairwise distances, same as var(*Y*) or trace(*K*):

$$\frac{1}{2N}\sum_{ij}\left|\vec{Y}_i - \vec{Y}_j\right|^2 = \sum_i \left|\vec{Y}_i\right|^2 = \sum_i K_{ii}$$

  (Similar intuition as PCA.)

- Boundedness

  Follows from triangle inequality and connectedness of neighborhood graph.

Semidefinite programming

Maximize trace(*K*) subject to:

(i) $K \geq 0,$

(ii) $\sum_{ij} K_{ij} = 0,$

(iii) for all neighborhoods (ijk),

$$K_{ii} - K_{ij} - K_{ik} + K_{jk}$$
$$= G_{ii} - G_{ij} - G_{ik} + G_{jk}$$

Convex optimization

- Solution

  Feasible region is convex.

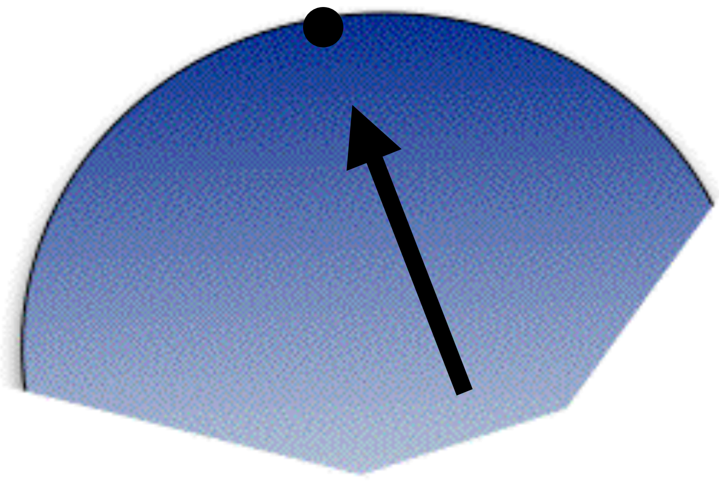
  Never empty (includes *G*).

  Objective is linear and bounded.

  Efficient algorithms exist.

- Caveat

  Generic solvers scale poorly.



Steps of SDE

1) K nearest neighbors

   Compute nearest neighbors, distances and angles.

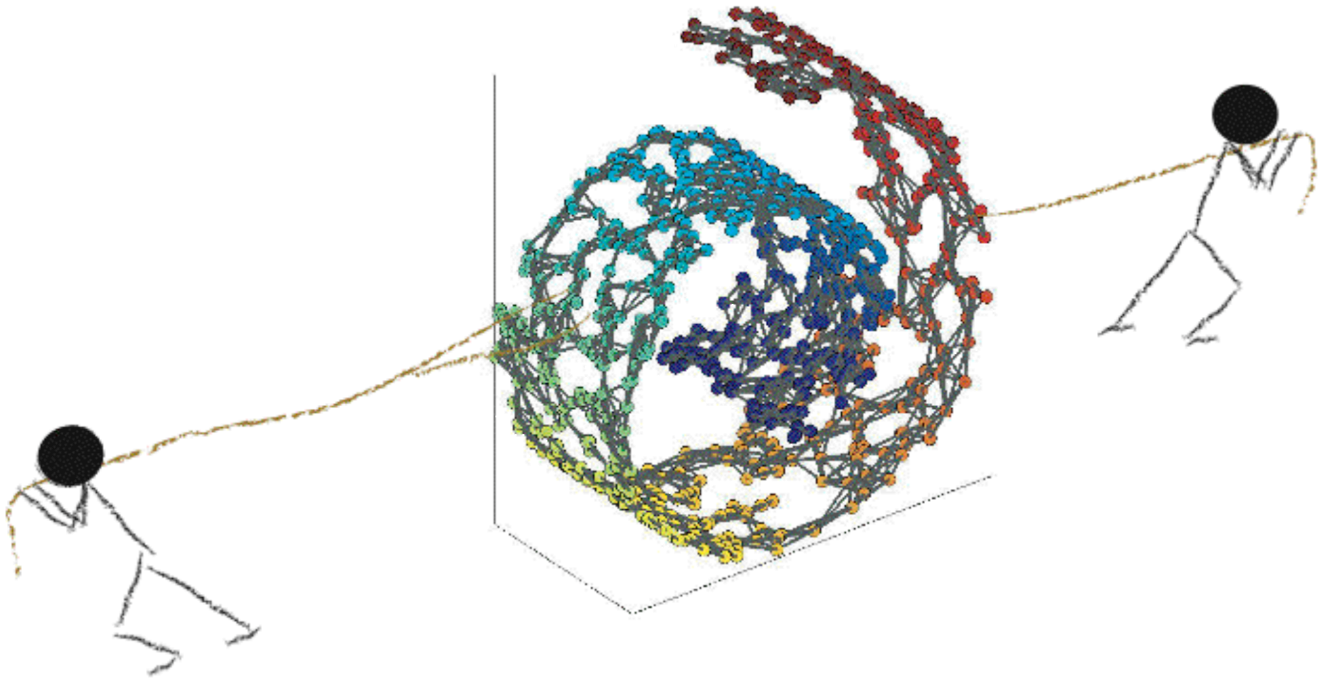
2) Semidefinite programming

   Maximize trace of centered, locally isometric Gram matrices.

3) Matrix diagonalization

   Top eigenvectors give embedding. Estimate *d* from eigenvalues.

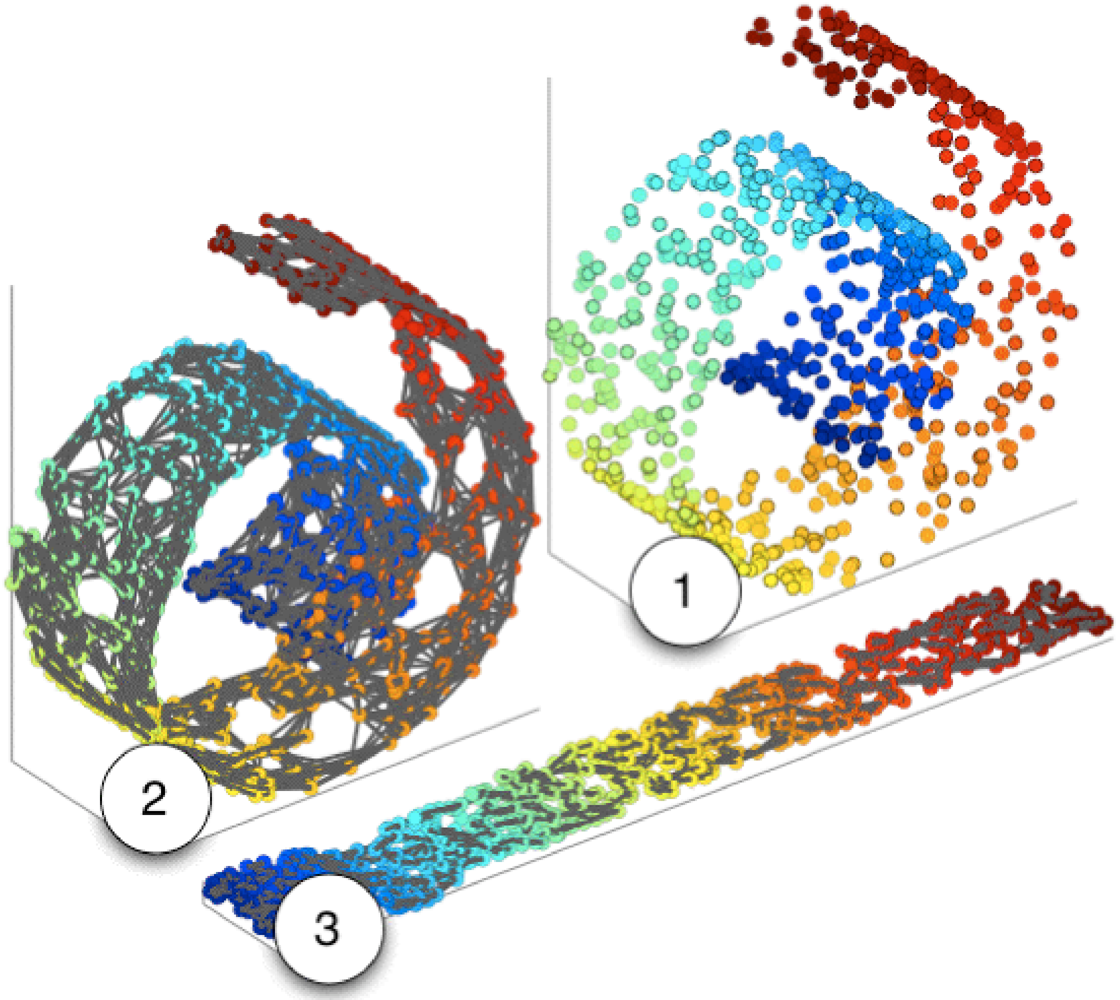
Experimental Results



“maximum variance unfolding”

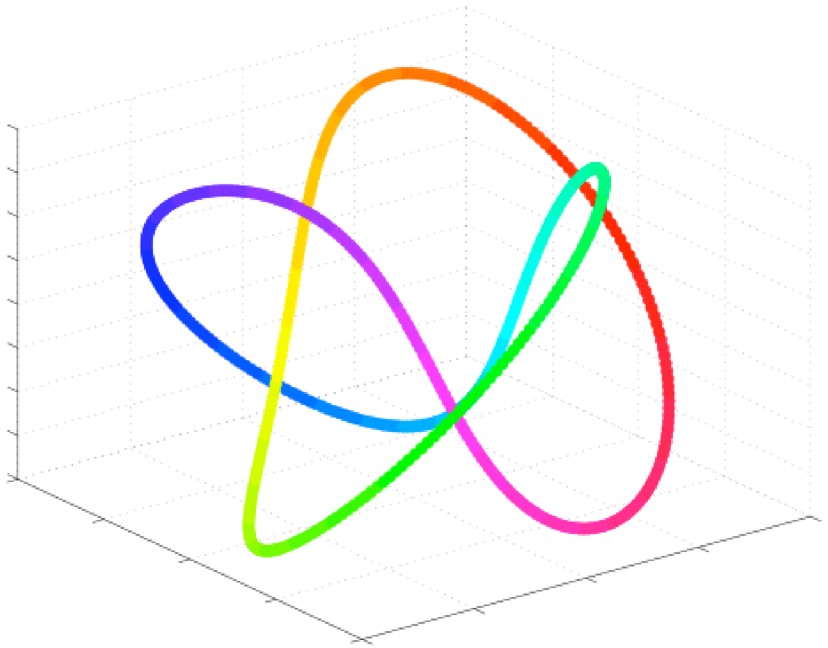
(Sun, Boyd, Xiao, & Diaconis)

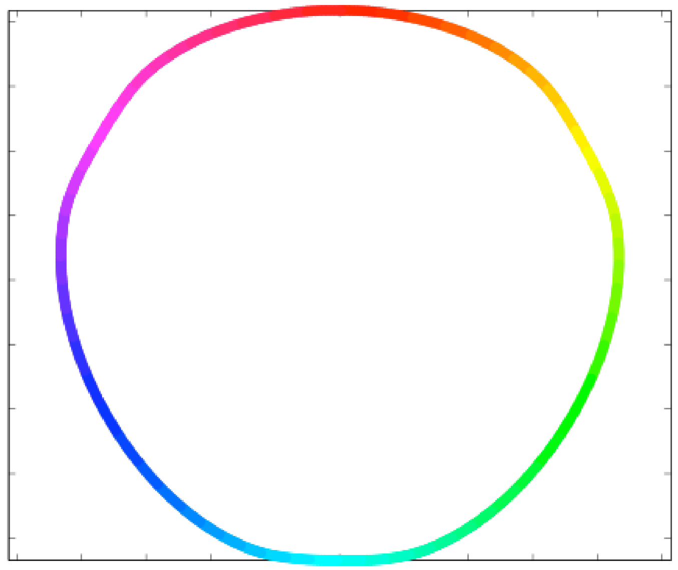
Swiss Roll



$$\boxed{\begin{aligned} N &= 800 \\ k &= 6 \end{aligned}}$$

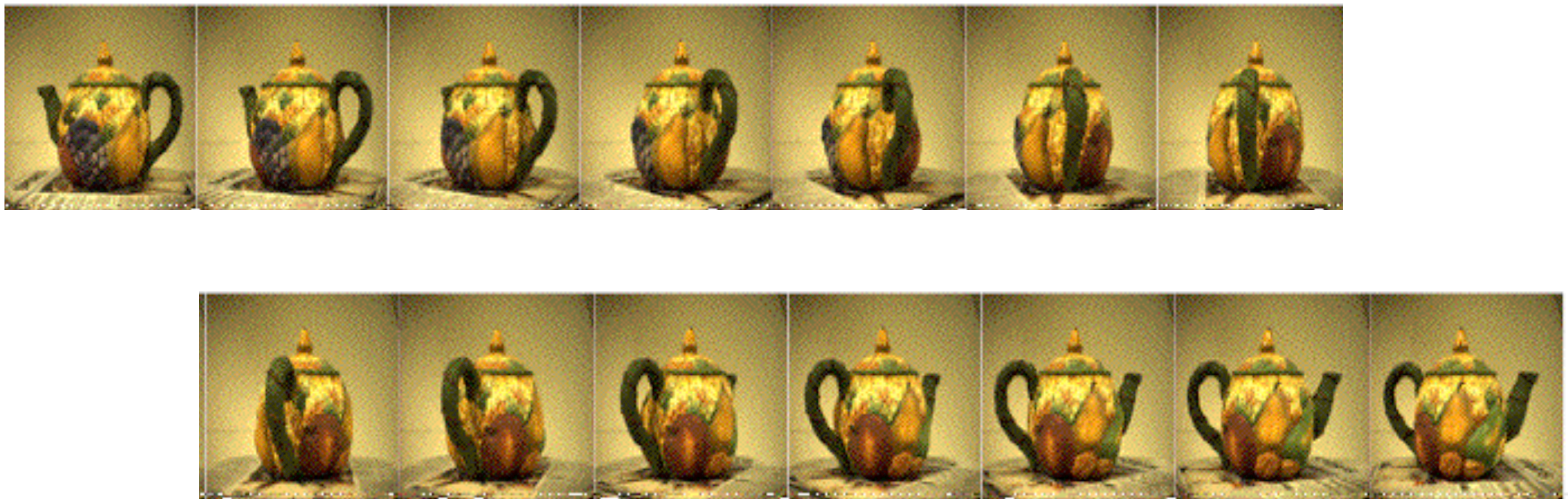
Trefoil knot





$$\boxed{\begin{aligned} N &= 539 \\ k &= 4 \end{aligned}}$$

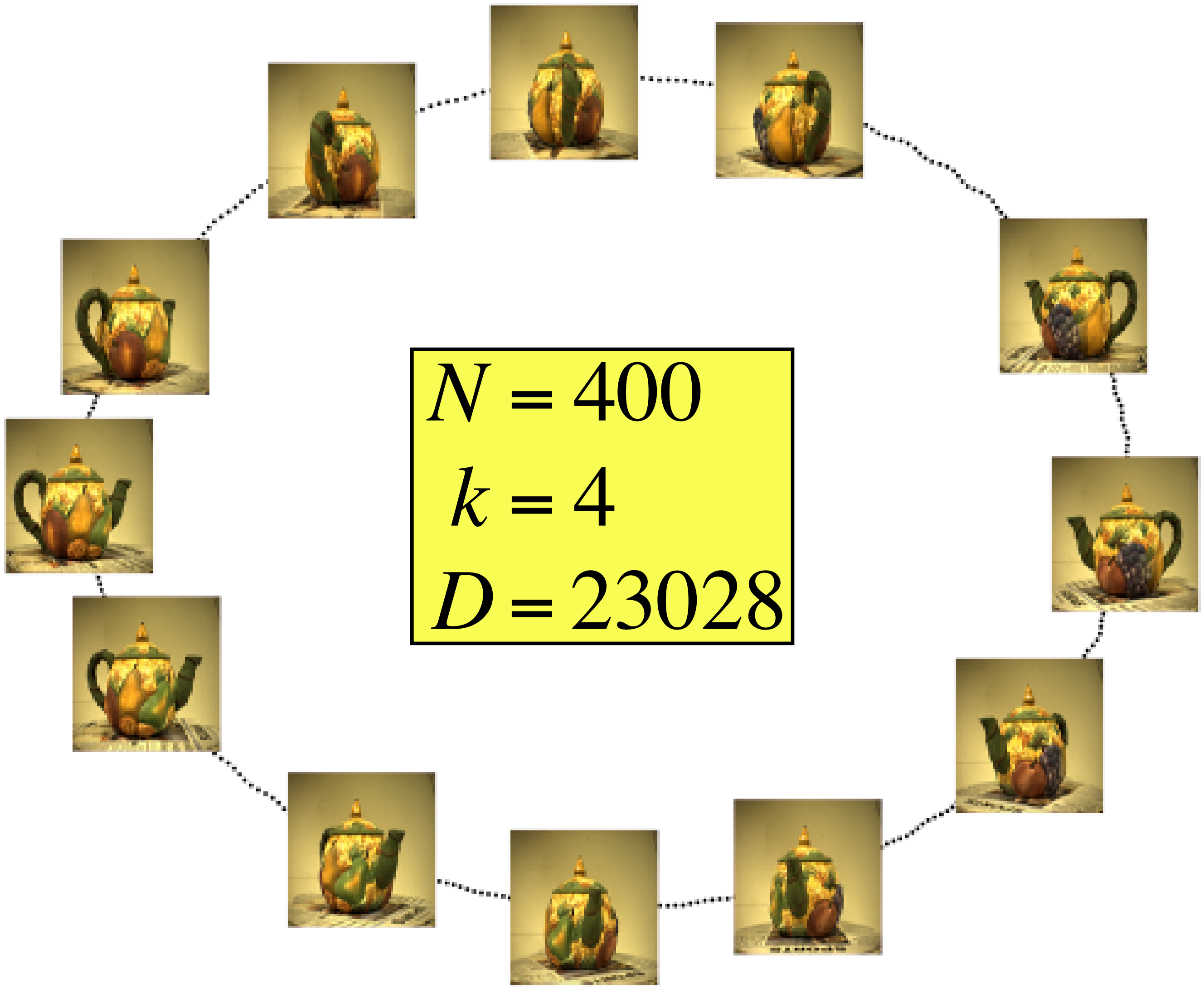
Teapot (half rotation)



Images ordered by one dimensional embedding

$N = 200$
$k = 4$
$D = 23028$

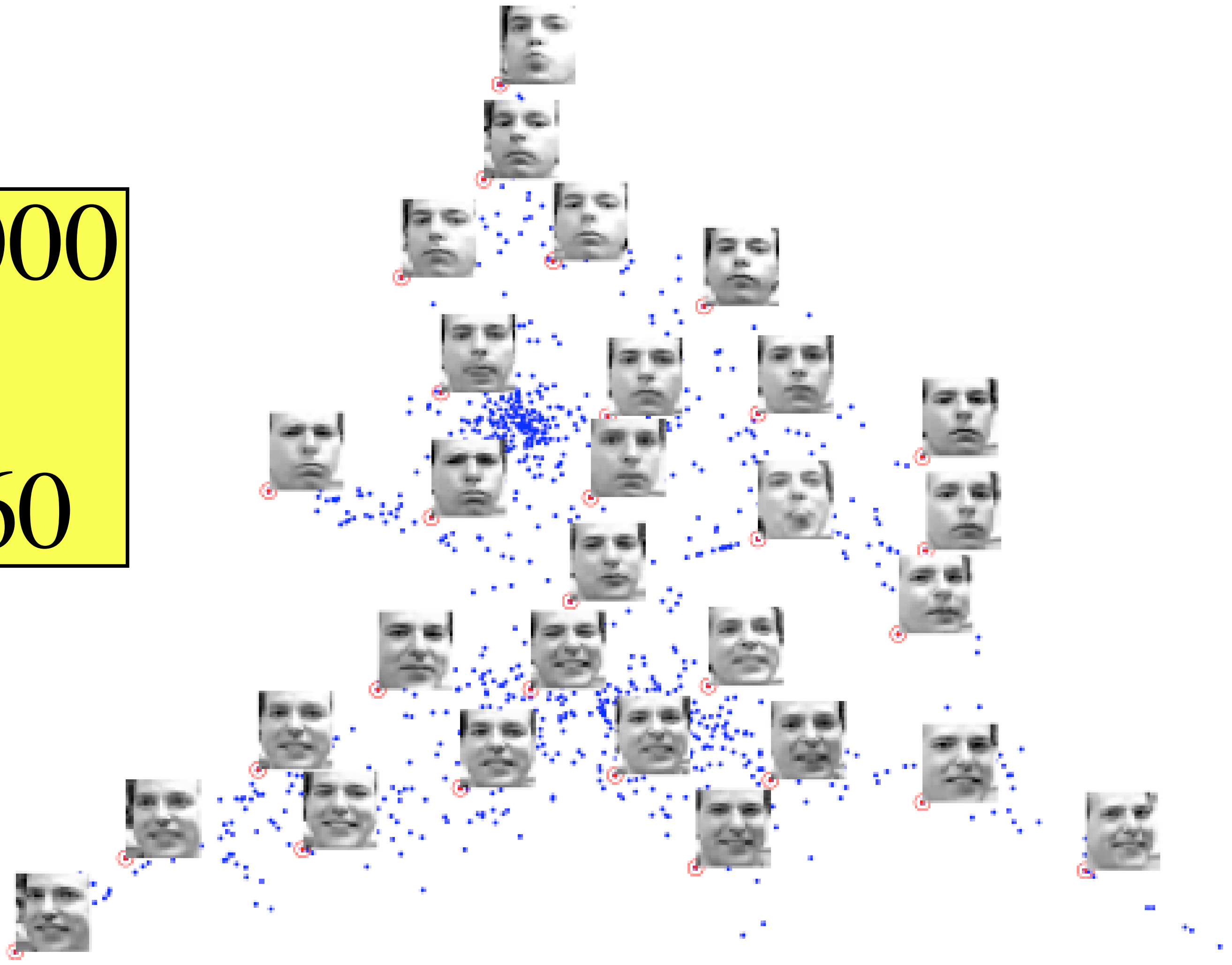
Teapot (full rotation)



$$N = 400$$
$$k = 4$$
$$D = 23028$$

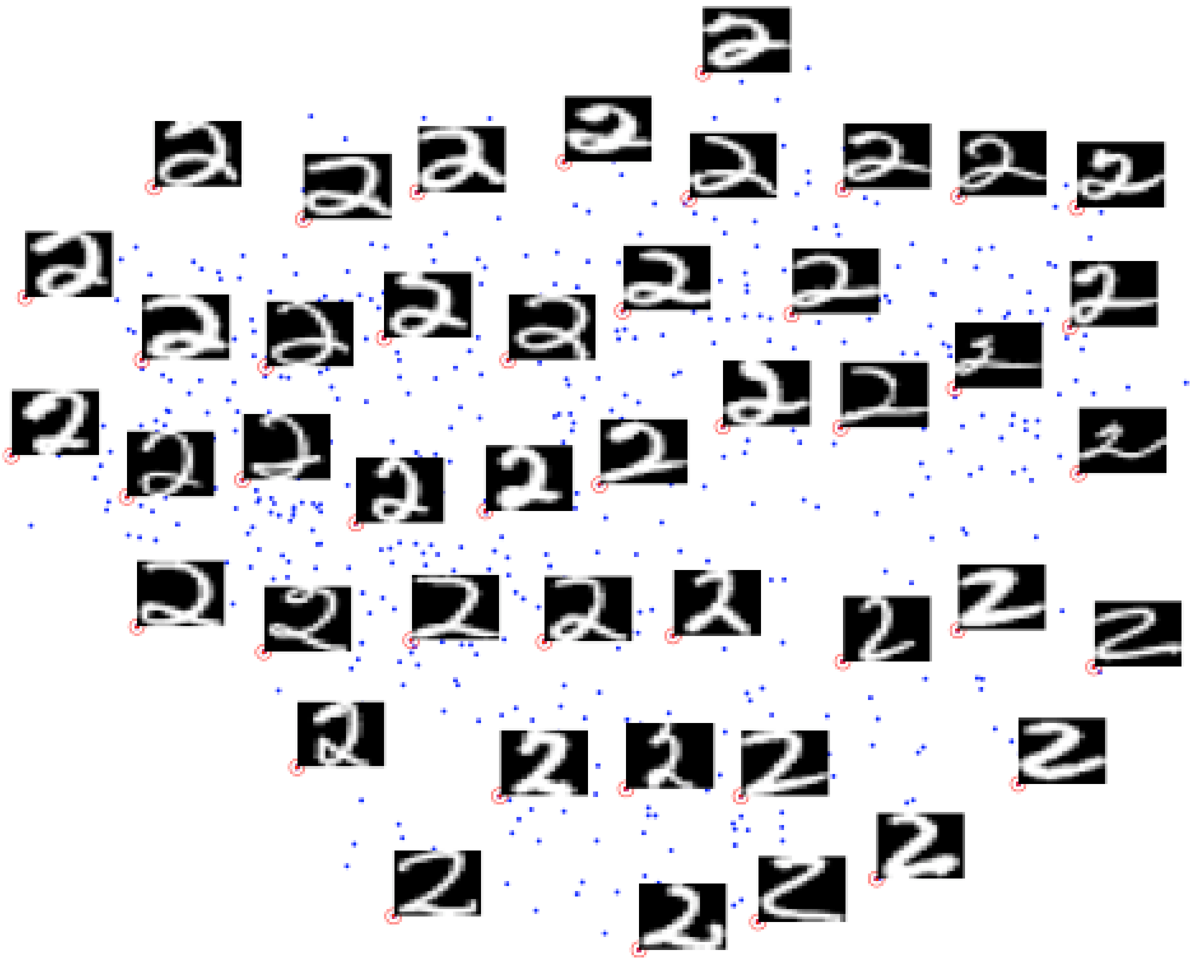
Images of faces

$$N = 1000$$
$$k = 4$$
$$D = 560$$

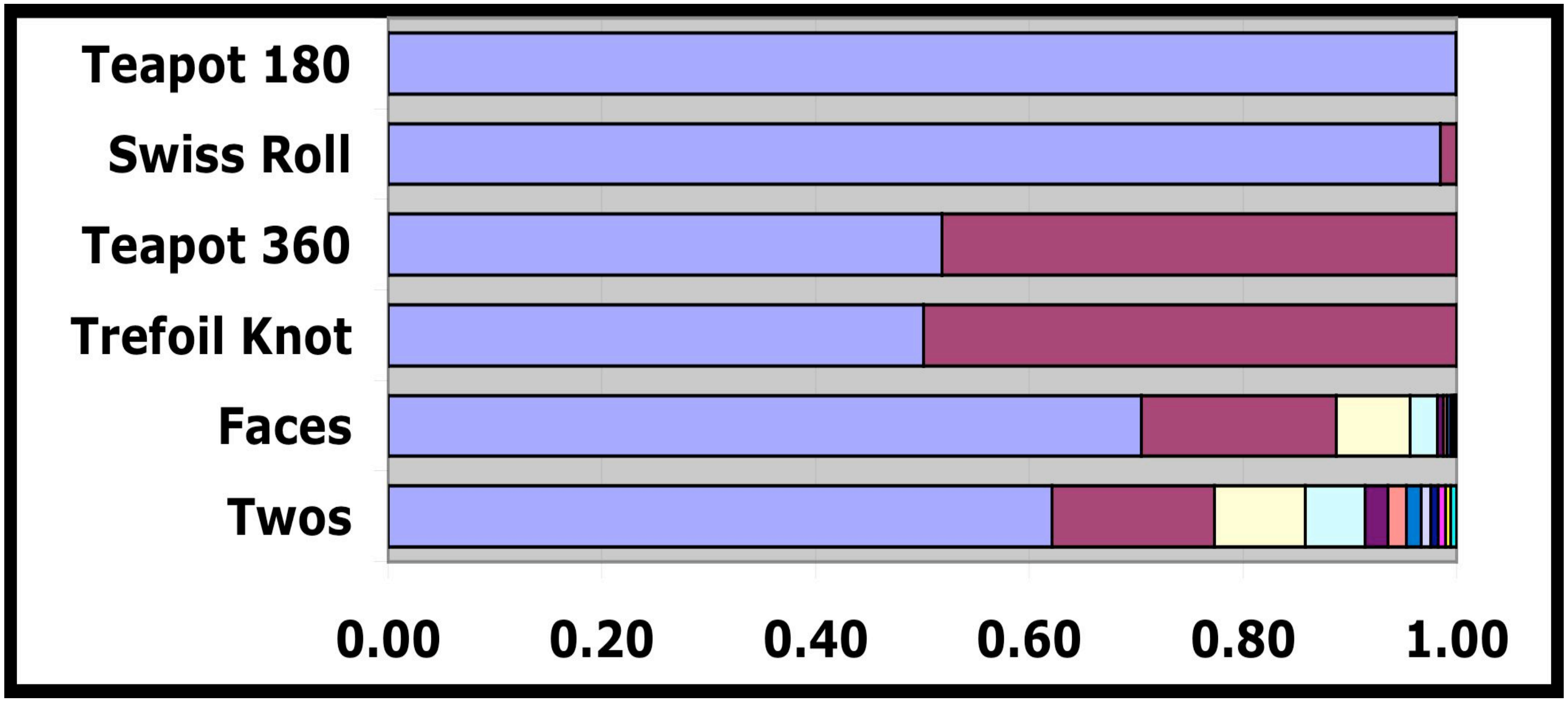


Handwritten digits

$$N = 638$$
$$k = 4$$
$$D = 256$$



Eigenvalues



(normalized by trace)

Evaluating SDE

- Pros
  - Eigenvalues reveal dimensionality.
  - Constraints ensure local isometry.
  - Algorithm tolerates small data sets.
- Cons
  - Computation intensive.
  - Currently limited to $N \leq 2000$, $k \leq 6$.

LLE vs SDE

- **Sparse vs dense**

  LLE constructs a sparse matrix.
  SDE constructs a dense matrix.

- **Bottom vs top**

  LLE computes bottom eigenvectors.
  SDE computes top eigenvectors.

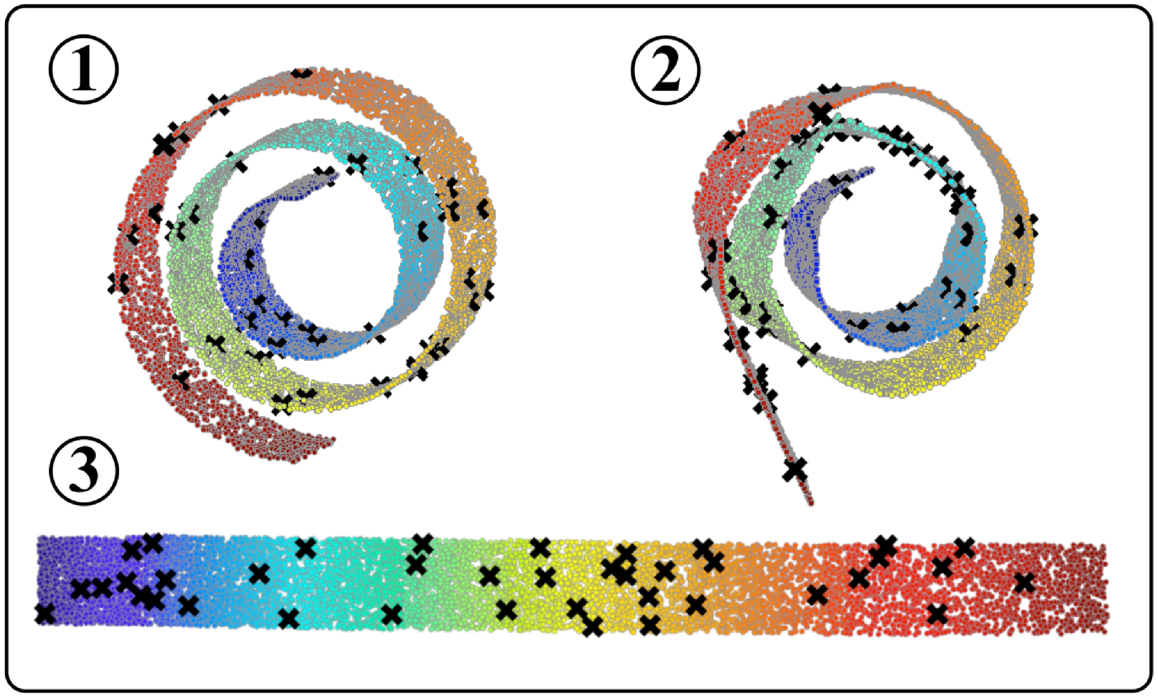
- **Estimating the dimensionality**

  LLE eigenvalues do not reveal *d*.
  SDE eigenvalues do reveal *d*.

Algorithm #3: ℓSDE

landmark SDE

(a happy marriage of LLE & SDE)



$N = 10000$
$k = 4$

Matrix factorization

- Why is SDE slow?

  Algorithm learns NxN matrix $K_{ij} = Y_i \bullet Y_j$.
  Solving SDPs is superlinear in N.

- Approximate $K \approx QLQ^T$

  Q is Nxn matrix (given).
  L is nxn matrix, with $n \ll N$ (learned).

Reformulation $K \approx QLQ^T$

- Old SDP over NxN matrix K

Maximize trace(K) **subject to:**

1) $K \succeq 0.$

2) $\Sigma_{ij} K_{ij} = 0.$

3) For all (i,j) **such that** $\eta_{ij}=1$,
$K_{ii} - 2K_{ij} + K_{jj} = \|\vec{x}_i - \vec{x}_j\|^2.$

- New SDP over nxn matrix L

Maximize trace(QLQ^T) **subject to:**

1) $L \succeq 0.$

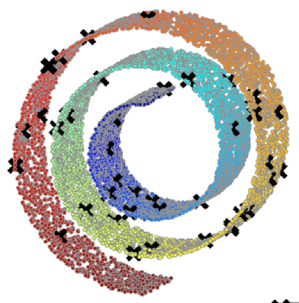
2) $\Sigma_{ij} (QLQ^T)_{ij} = 0.$

3) For all (i,j) **such that** $\eta_{ij}=1$,
$(QLQ^T)_{ii} - 2(QLQ^T)_{ij} + (QLQ^T)_{jj} \leq \|\vec{x}_i - \vec{x}_j\|^2.$

Sketch of idea

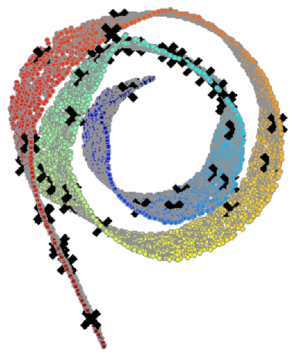
- Choose landmarks:

$$\boxed{\{\vec{\ell}_\alpha\}_{\alpha=1}^{n} \text{ where } n << N}$$



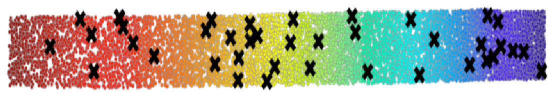
- Reconstruct inputs:

$$\boxed{\vec{x}_i \approx \hat{x}_i = \sum_\alpha Q_{i\alpha}\vec{\ell}_\alpha}$$



- Unfold inputs:

$$\boxed{\vec{y}_i \approx \hat{y}_i = \sum_\alpha Q_{i\alpha}\vec{\ell}_\alpha}$$



- Matrix factorization

$$\boxed{\vec{y}_i \cdot \vec{y}_j \approx QLQ^T \text{ with } L_{\alpha\beta} = \vec{\ell}_\alpha \cdot \vec{\ell}_\beta}$$

Reconstructing from landmarks

- Error function

$$\boxed{\Phi(W,X) = \sum_i \left|\vec{X}_i - \sum_j W_{ij}\vec{X}_j\right|^2}$$

- Optimizations

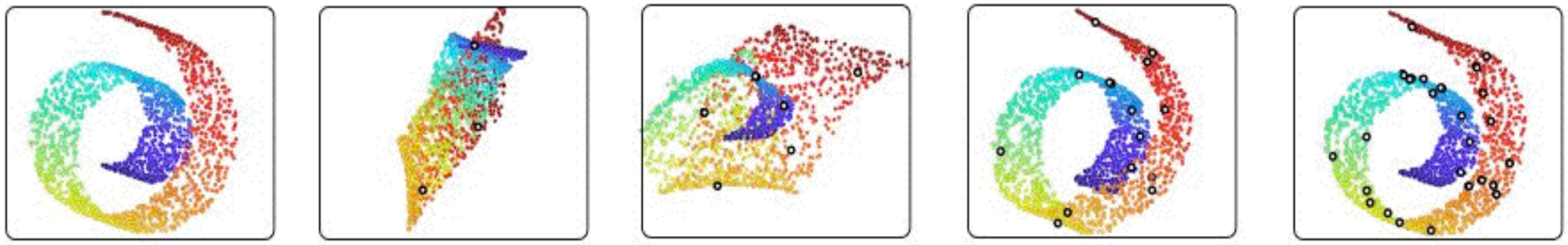
  Compute weights W_{ij} as in LLE.
  Clamp landmarks; reconstruct inputs.

$$\boxed{\hat{x}_i = \min_{x\,\Box\,\Box}\left[\Phi(W,X)\right] = \sum_{\alpha} Q_{i\alpha}\vec{\ell}_{\alpha}}$$

  Reconstruct by solving a sparse system of linear equations.

Reconstructing from landmarks

- Input reconstructions



N=2000 *n=4* *n=8* *n=16* *n=32*

- Output reconstructions

LLE weights are invariant to unfolding.
Same matrix reconstructs outputs!

$$\hat{x}_i = \min_{x \notin \ell} \left[\Phi(W,X)\right] = \sum_{\alpha} Q_{i\alpha} \vec{\ell}_{\alpha}$$

$$\hat{y}_i = \min_{y \notin \ell} \left[\Phi(W,Y)\right] = \sum_{\alpha} Q_{i\alpha} \vec{\ell}_{\alpha}$$

Steps of ℓSDE

As in LLE:

(1) Compute nearest neighbors.

(2) Compute LLE weights W.

(3) Choose landmarks.

(4) Compute landmark weights Q.

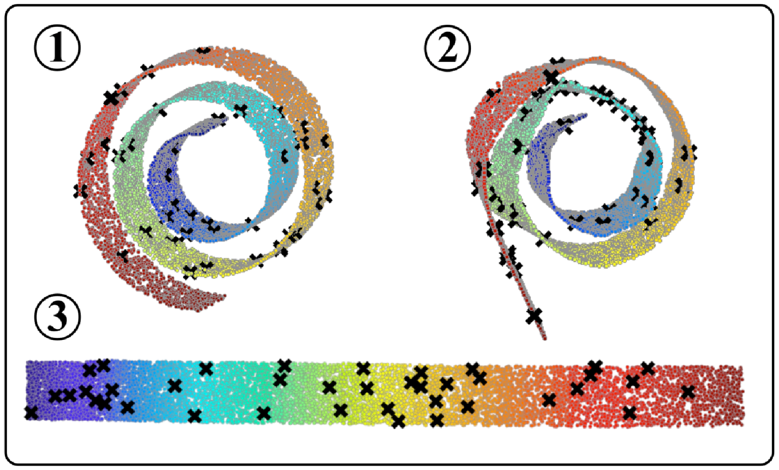
As in SDE:

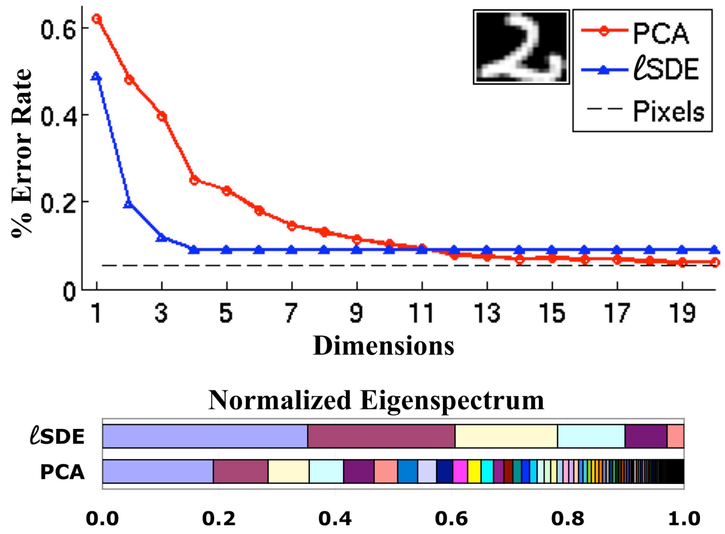
(5) Solve SDP to unfold landmarks.

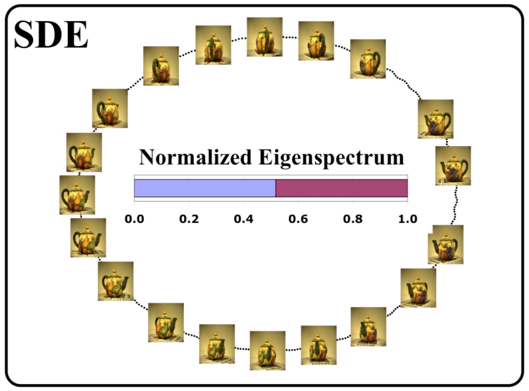
(6) Compute top eigenvectors.

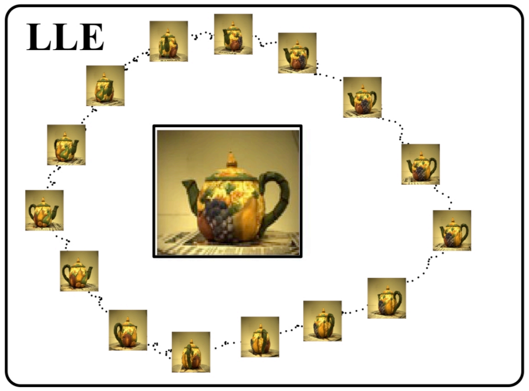
(7) Construct outputs from landmarks.

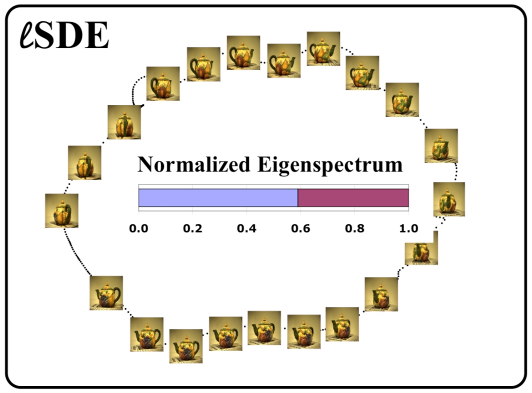
Experimental results



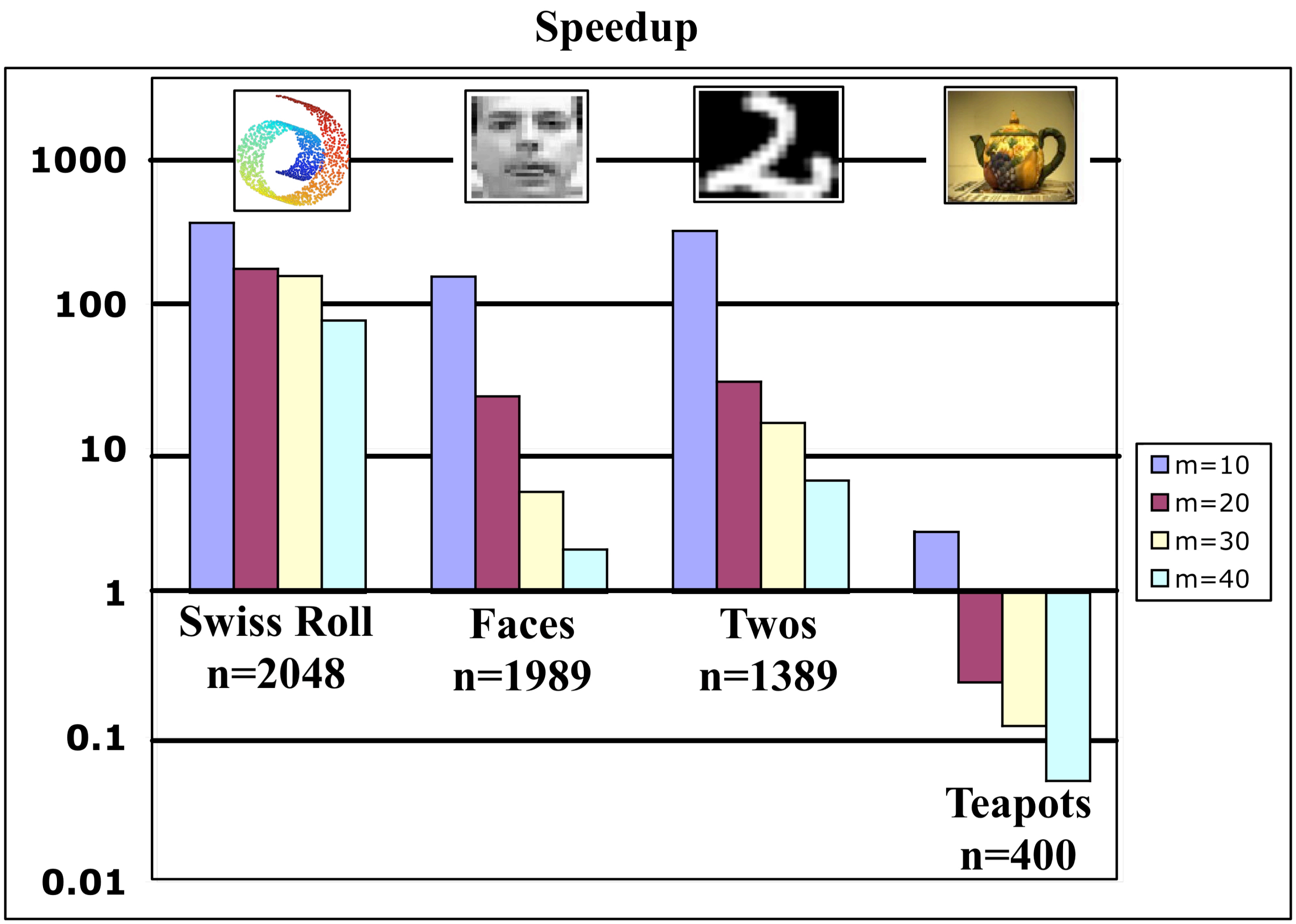








How much faster?



Related work

- Other algorithms:

  Isomap, Laplacian eigenmaps, local tangent space alignment, hessian LLE, charting

- Common framework:

  1) Compute nearest neighbors.
  2) Construct an N x N matrix.
  3) Compute eigenvectors.

“Local” vs “global” methods

- Local methods (LLE, LTSA, …)

  Construct sparse matrix.
  Compute bottom eigenvectors.
  Scale (relatively) well.

- Global methods (Isomap, SDE)

  Construct dense matrix.
  Compute top eigenvectors.
  Eigenvalues reveal dimensionality.

Landmark methods

- ℓIsomap

  Distances to landmarks are used to "triangulate" non-landmarks.

- ℓSDE

  Landmark locations are propagated through sparse weighted graph.

  Analogous to recent work in semi-supervised learning.

  (Belkin, Matveeva, & Niyogi; Smola & Kondor; Zhu, Ghahramani, & Lafferty)

Conclusion

- Big ideas
  - Manifolds are everywhere.
  - Graph-based methods can learn them.
- Ongoing work
  - Scaling up to larger data sets
  - Theoretical guarantees
  - Alternative topologies
  - Extrapolation and functional maps