CS598PS – Machine Learning for Signal Processing

Dimensionality Reduction – Non-linear Approaches

20 September 2017

Today's lecture

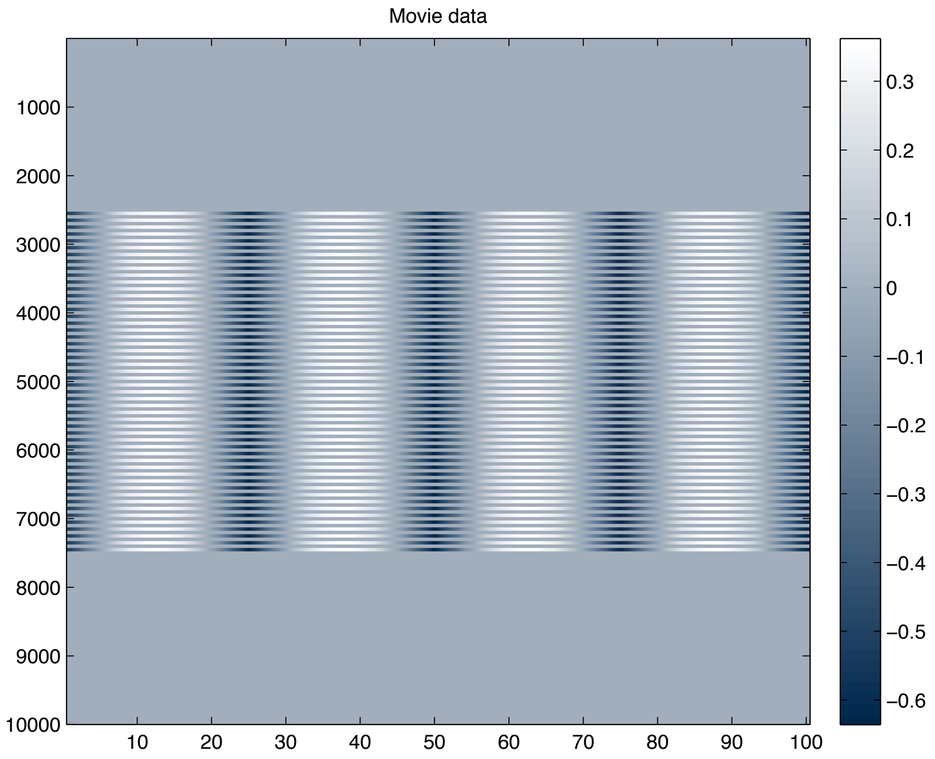
- Non-linear dimensionality reduction
- Kernel PCA
- Manifold Methods

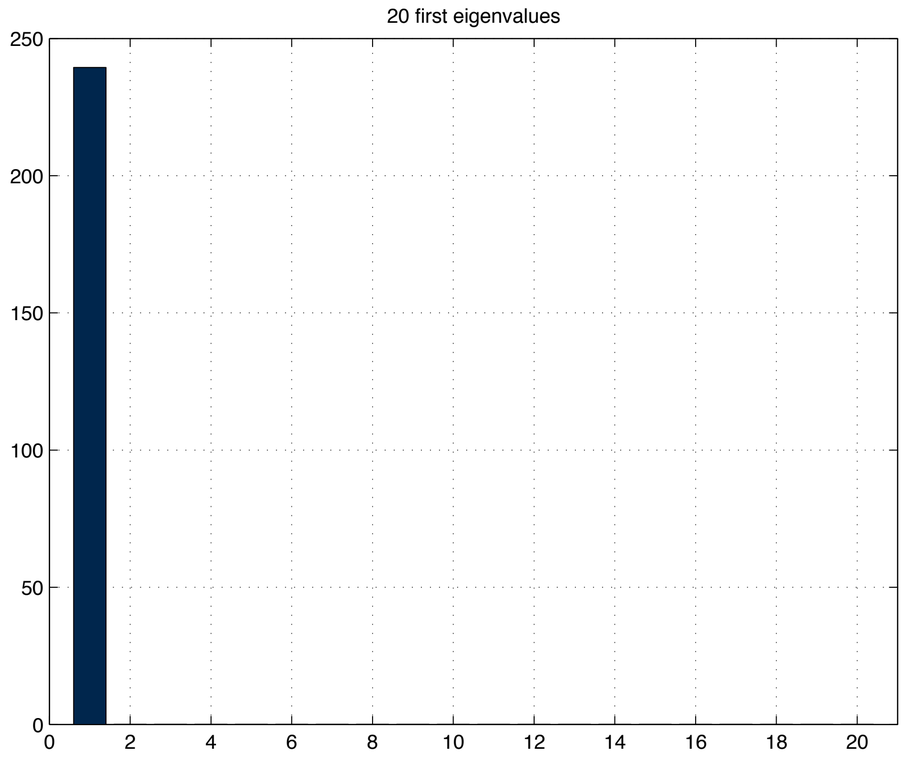
Why dimensionality reduction

- Data can be hugely redundant
  - What are the (data) dimensions of this 100×100×100 video?

Very low dimensionality!

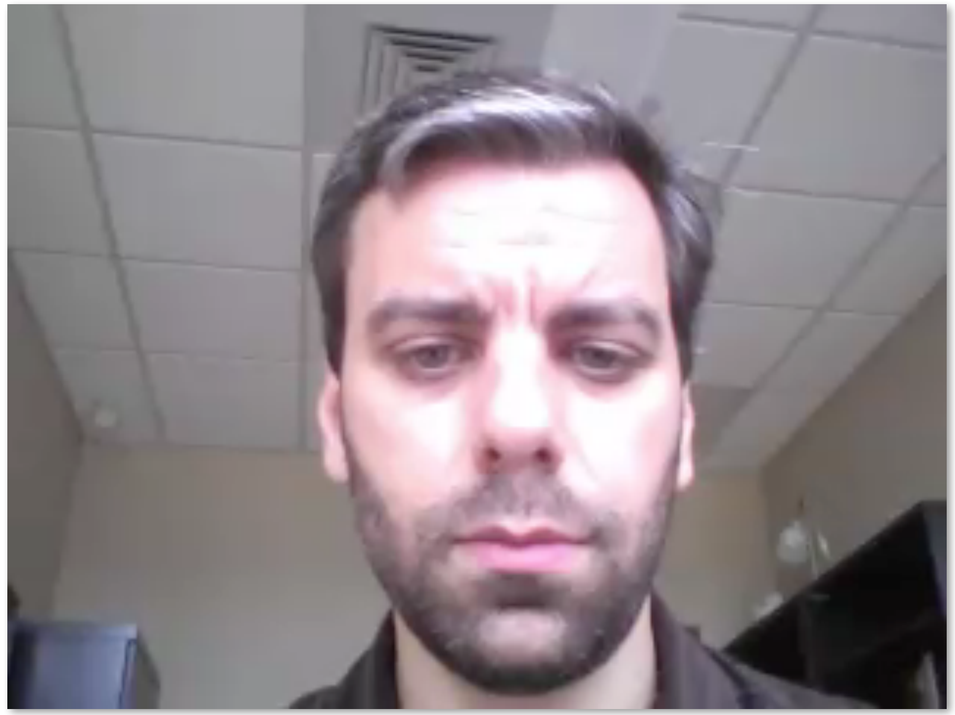
- We effectively need only one pixel to express the video
  - All other active pixels behave the same





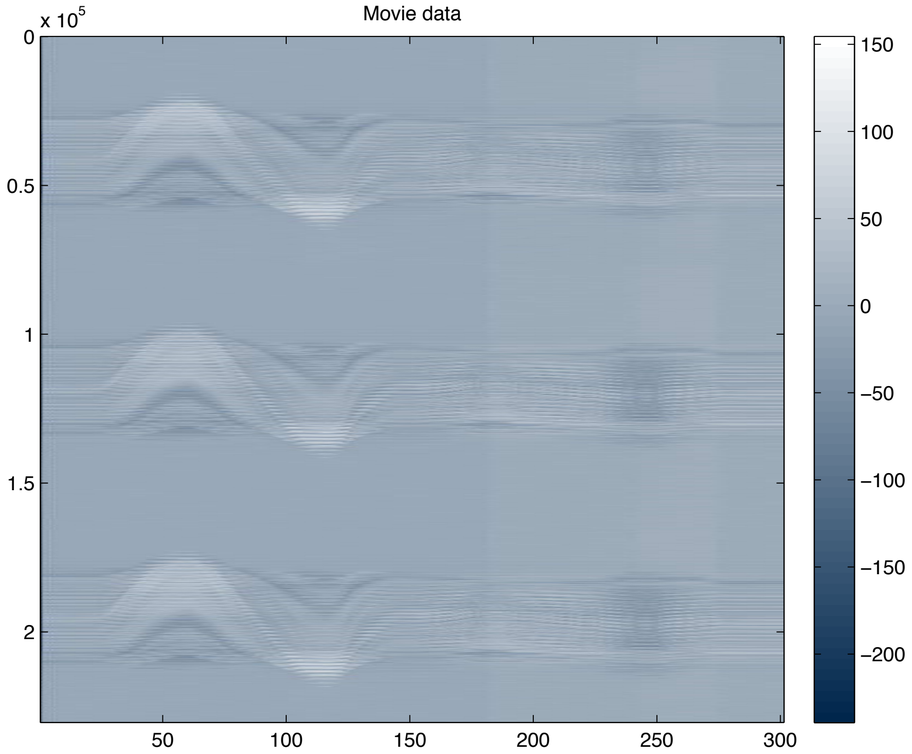
In real life though …

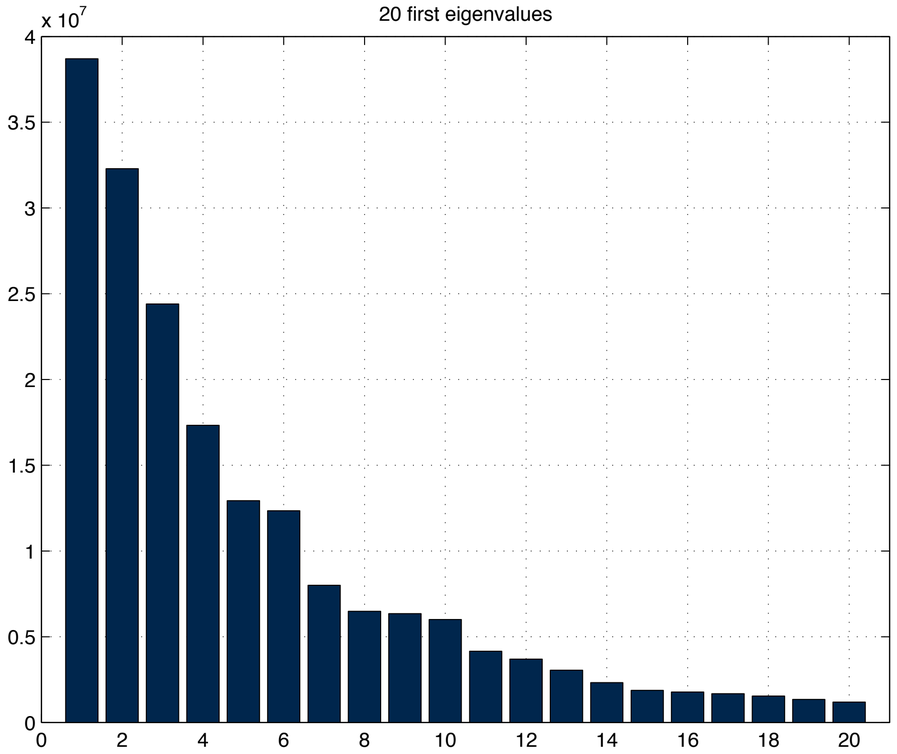
- What is the dimensionality now?
  - Lots of data, $320 \times 240 \times 3 \times 337$



It looks high dimensional

- Less than the data implies, but more than what we see
  - I still think it is around 4-dimensional, though



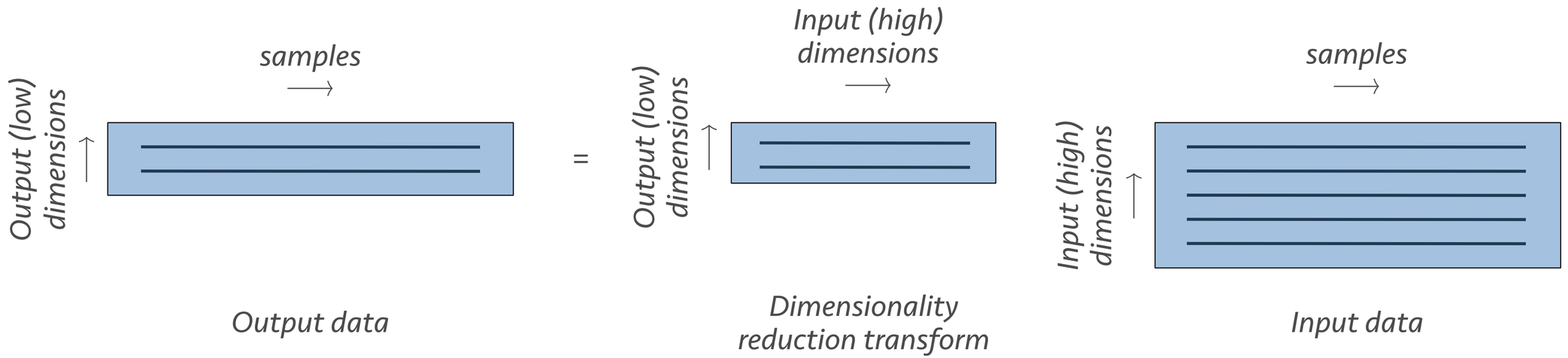


Linear dimensionality reduction

- Linear transform to drop dimensions

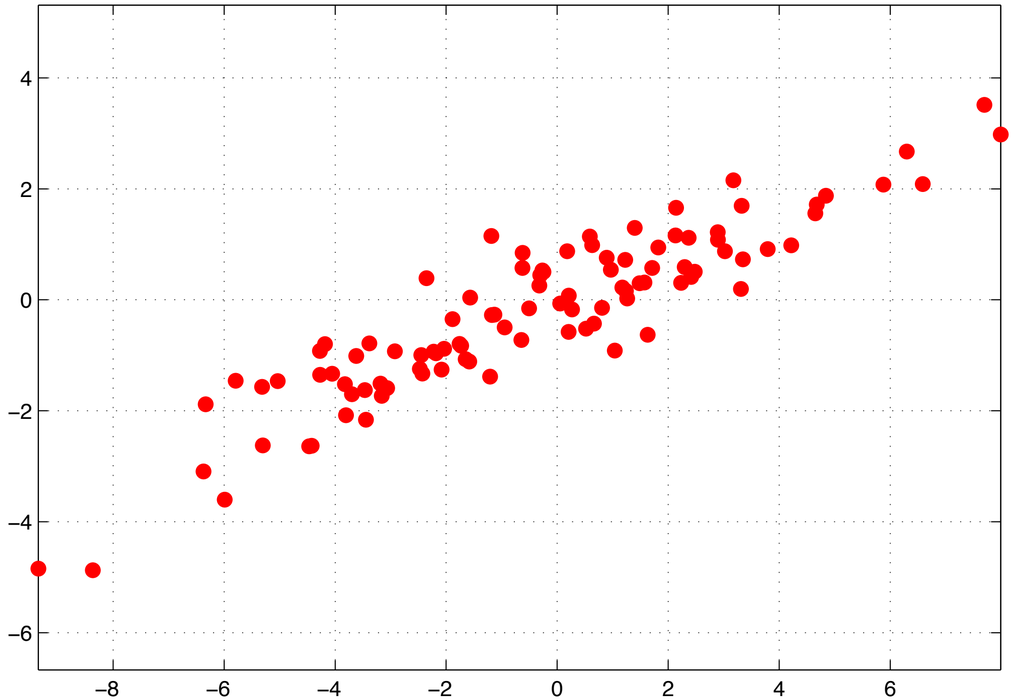
$$\mathbf{Y} = \mathbf{W} \cdot \mathbf{X}$$

  - We saw this with PCA/NMF



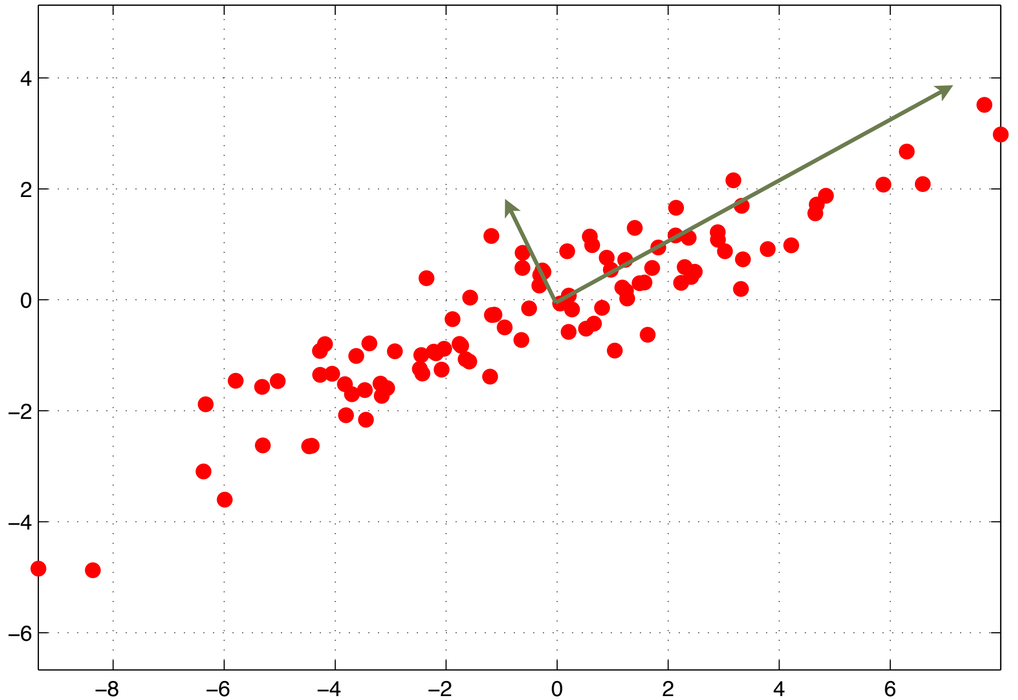
"Easy" data

- Linear transforms work fine with simple data



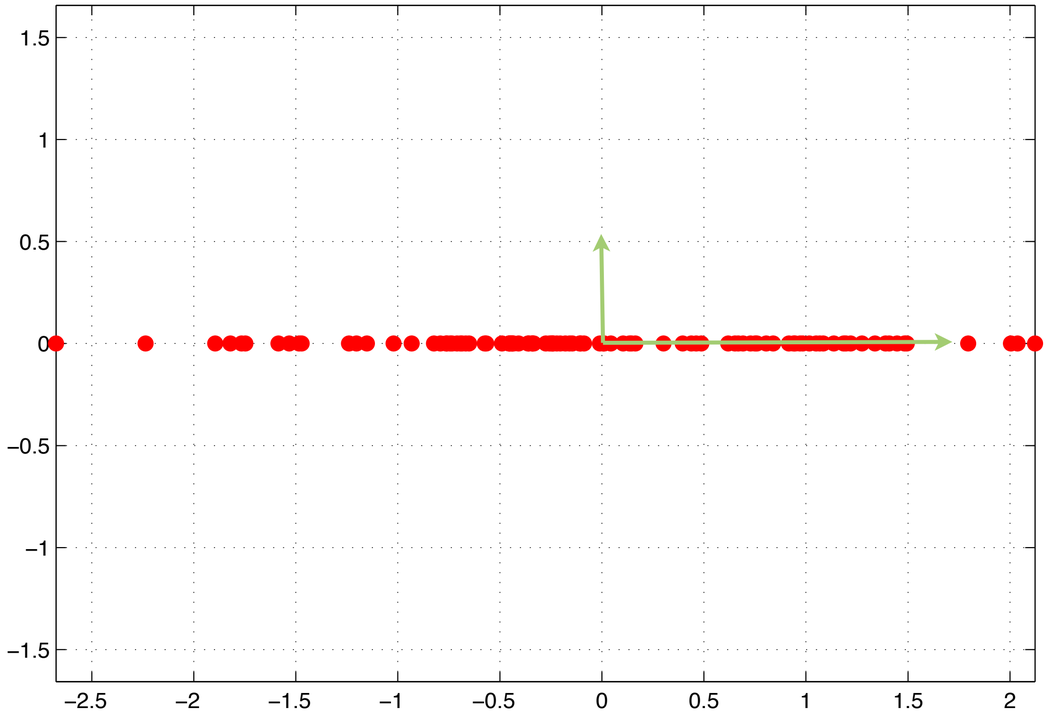
“Easy” data

- Principal components show directions of maximal variance



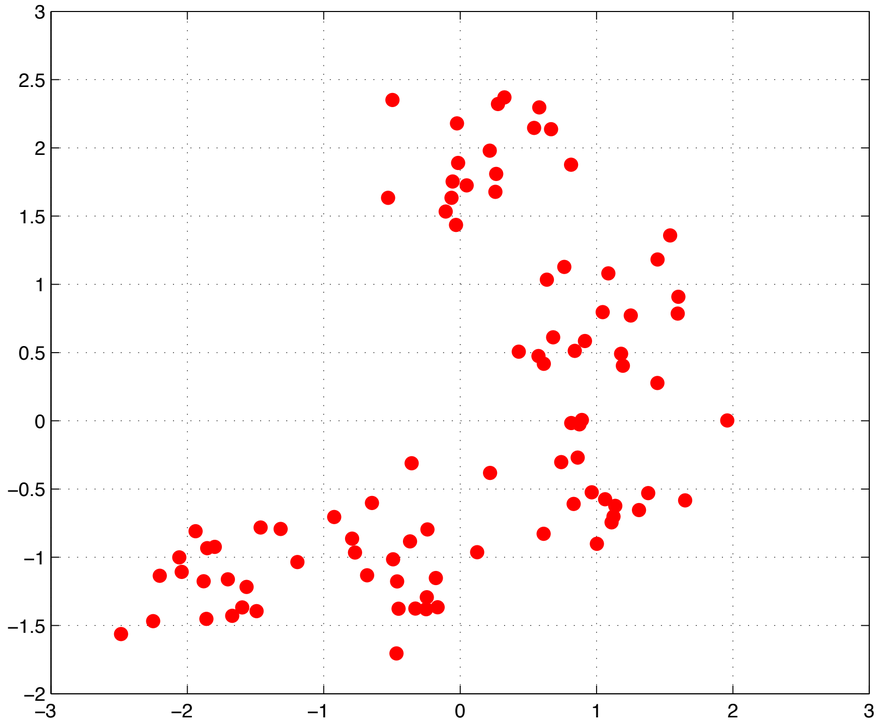
"Easy" data

- Keep only the large dimensions to reduce dimensionality



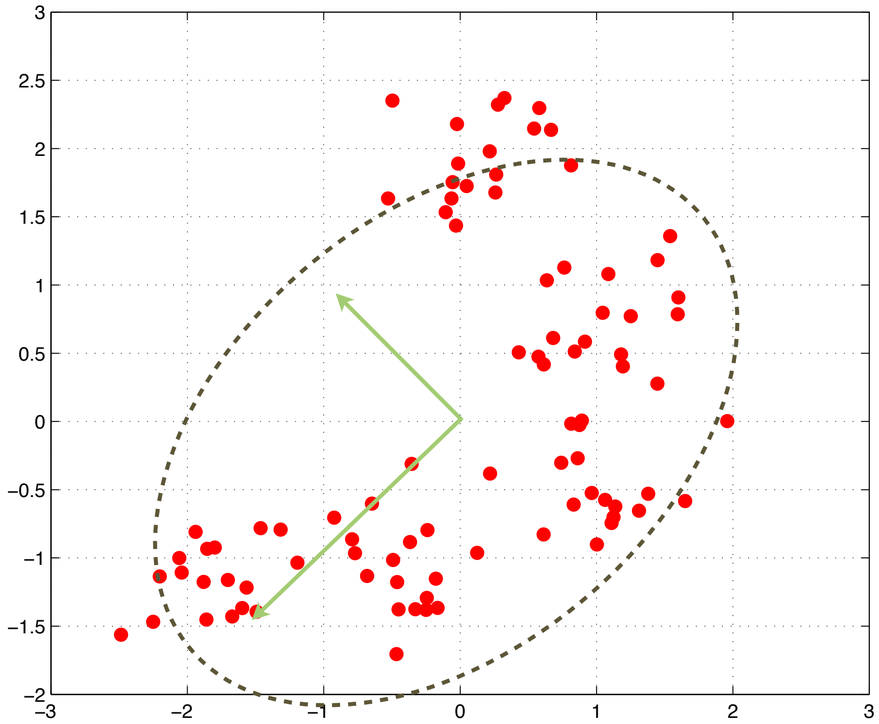
More complicated data

- If data isn't Gaussian(ish) PCA doesn't help



More complicated data

- Not much to interpret here …

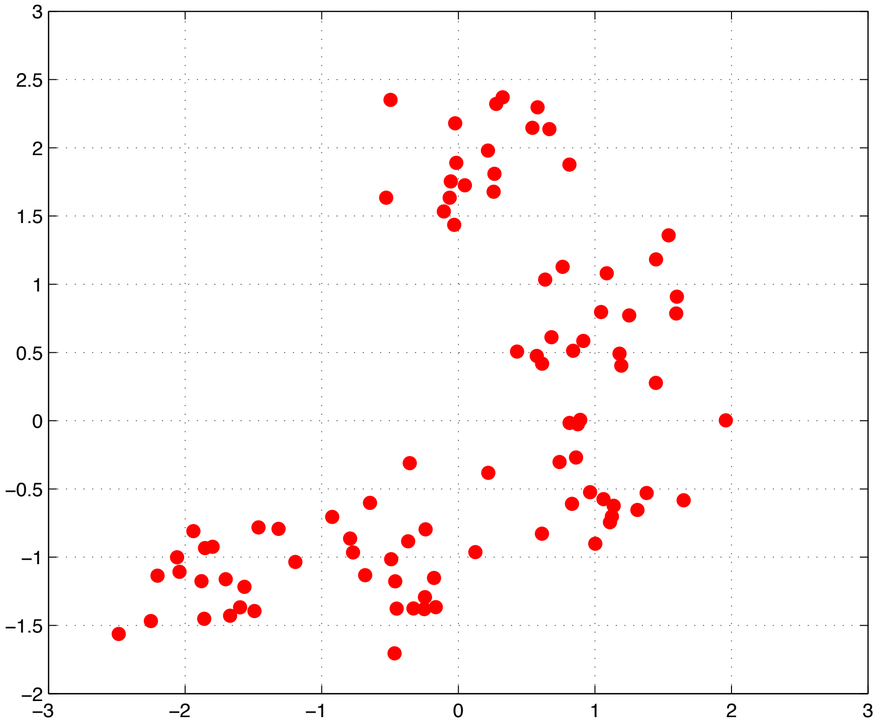


Pinpointing the problem

- The principal components are linear
- The data we observe will not always conform to that
- Can we define “non-linear” components?

Going the other way

- Suppose that we have some "curvy" data
- There should be a non-linear mapping that "straightens" that data out



Going the wrong way

- With N, D-dimensional data points

$$\mathbf{x}_n \in \mathbb{R}^D, n = \{1, \ldots, N\}$$

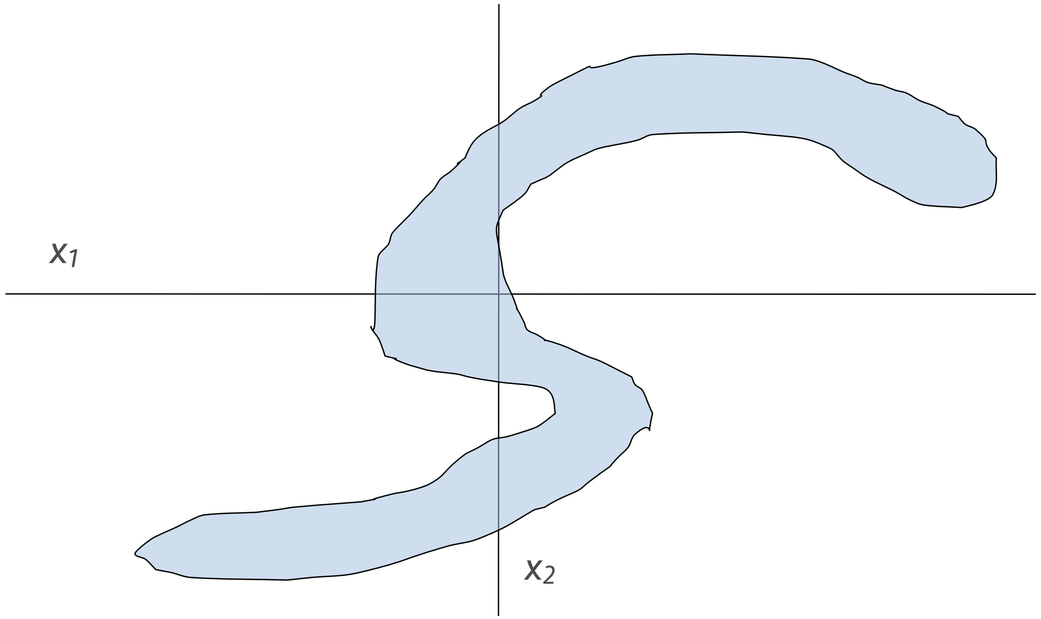
- Assume an unspecified non-linear mapping

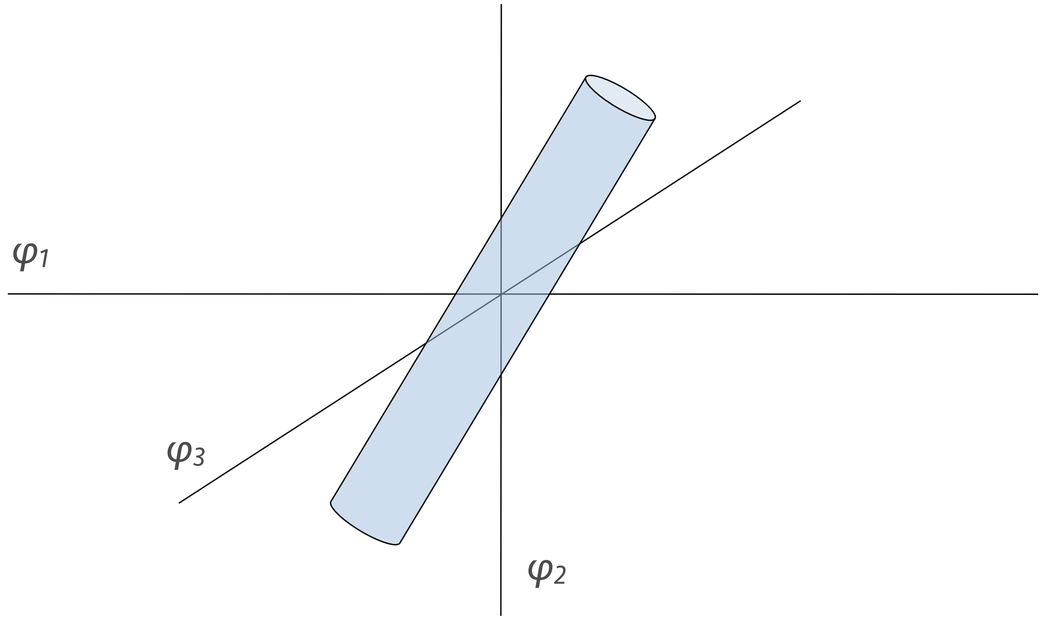
$$\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M, \mathbf{x} \mapsto \mathbf{z} = \phi(\mathbf{x}), M > D$$

  - i.e. we non-linearize and increase the dimension of our data!

In pictures

$$\mathbf{x} \in \mathbb{R}^2 \rightarrow \phi(\mathbf{x}) \in \mathbb{R}^3$$





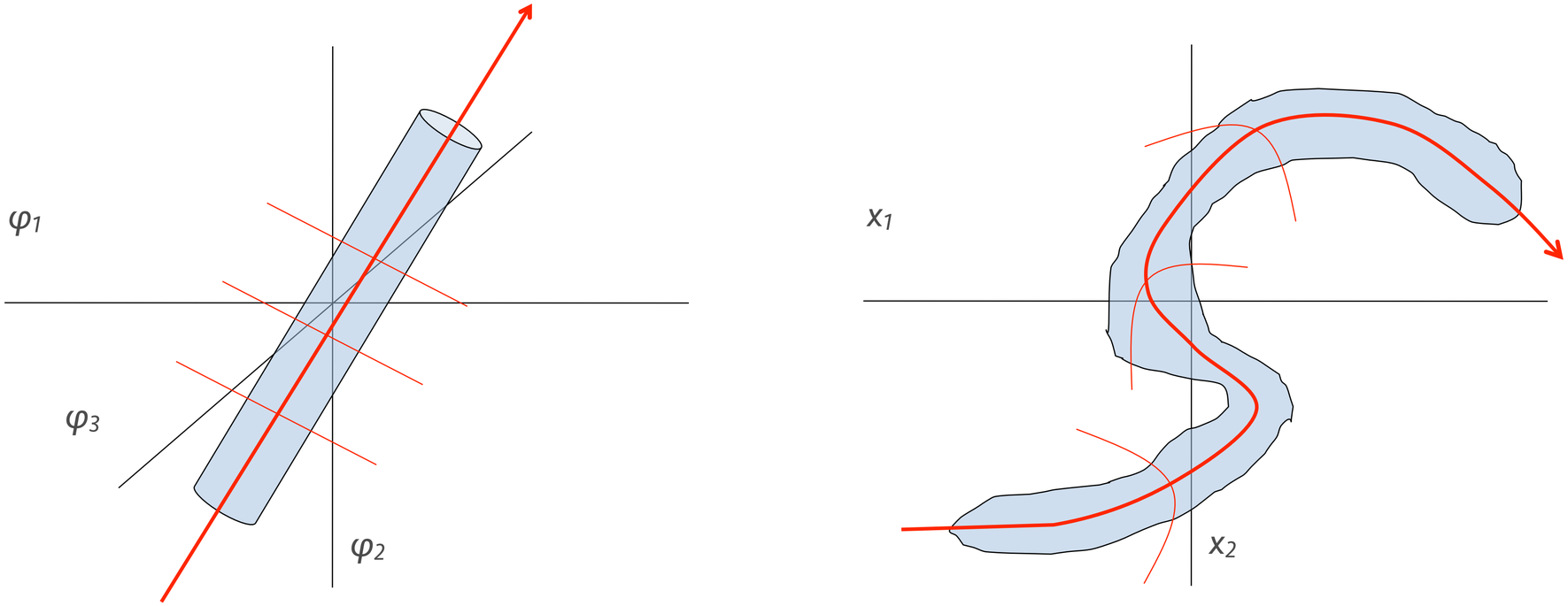
Now we can do PCA!

- With the right mapping, the new data can be well-behaved for PCA

- The principal components will be in the (potentially higher) M dimensional space

Going back to the original space

$$\phi(\mathbf{x}) \in \mathbb{R}^3 \rightarrow \mathbf{x} \in \mathbb{R}^2$$



From PCA to kernel PCA

- In regular PCA we do:

$$\mathbf{S} = \mathrm{Cov}(\mathbf{x})$$ ← *Data covariance*

$$\mathbf{S} \cdot \mathbf{u}_i = \lambda_i \mathbf{u}_i$$ ← *Eigendecomposition*

- In kernel PCA we want to do:

$$\mathbf{C} = \mathrm{Cov}\Big(\phi(\mathbf{x})\Big)$$ ← *Transformed data covariance*

$$\mathbf{C} \cdot \mathbf{v}_i = \lambda_i \mathbf{v}_i$$ ← *Eigendecomposition*

Some problems

- How do we choose the map?
  - We have to operate in this new domain now
- We are in potentially more dimensions
  - More computations == slower!

Some reshuffling

- From the definition of the eigendecomposition

$$\mathbf{C} \cdot \mathbf{v}_i = \lambda_i \mathbf{v}_i$$

- Each eigenvector v_i can be described by a linear combination of the input data, so we rewrite the above:

$$\mathbf{v}_i = \sum_n a_{i,n} \phi(\mathbf{x}_n)$$

$$\mathbf{C} \cdot \mathbf{v}_i = \lambda_i \mathbf{v}_i \Rightarrow \frac{1}{N} \sum_n \phi(x_n) \phi(x_n)^\top \cdot \left(\sum_m a_{im} \phi(x_m) \right) = \lambda_i \left(\sum_m a_{im} \phi(x_m) \right)$$

The kernel trick

- Under certain constraints (more later)

$$\phi(\mathbf{x})^\top \cdot \phi(\mathbf{y}) = \mathrm{K}(\mathbf{x}, \mathbf{y})$$

  - Where $\mathrm{K}()$ is easier to compute, e.g.:

4-dimensional map

$$\mathbf{x} = \left[x_1,\ x_2\right]^\top \in \mathbb{R}^2, \quad \phi\left(\mathbf{x}\right) = \left[x_1{}^2,\ x_1x_2,\ x_2x_1,\ x_2^2\right]^\top \in \mathbb{R}^4$$

$$\phi\left(\mathbf{x}\right)^\top \cdot \phi\left(\mathbf{y}\right) = \left[x_1{}^2,\ x_1x_2,\ x_2x_1,\ x_2^2\right] \cdot \left[y_1{}^2,\ y_1y_2,\ y_2y_1,\ y_2^2\right]^\top =$$

$$= \left(\left[x_1,\ x_2\right] \cdot \left[y_1,\ y_2\right]^\top\right)^2 = K\left(\mathbf{x}, \mathbf{y}\right)$$

2-dimensional computation!

Rewriting the analysis

- The starting high-dim eigendecomposition was:

$$\mathbf{C}\cdot\mathbf{v}_i=\lambda_i\mathbf{v}_i\Rightarrow\frac{1}{N}\sum_n\phi(x_n)\phi(x_n)^\top\cdot\left(\sum_m a_{im}\phi(x_m)\right)=\lambda_i\left(\sum_m a_{im}\phi(x_m)\right)$$

- We can now rewrite as a lower-dim version:
  - Multiply on both sides with $\phi(\mathbf{x}_l)$

$$\frac{1}{N}\sum_n \mathrm{K}(\mathbf{x}_l,\mathbf{x}_n)\sum_m a_{im}\,\mathrm{K}(\mathbf{x}_l,\mathbf{x}_m)=\lambda_i\sum_m a_{im}\,\mathrm{K}(\mathbf{x}_l,\mathbf{x}_m)$$

$$\Rightarrow\mathbf{K}^2\cdot\mathbf{a}_i=\lambda_i N\mathbf{K}\cdot\mathbf{a}_i$$

That's solved easily

- It is an eigendecomposition once again!

$$\mathbf{K} \cdot \mathbf{a}_i = \lambda_i N \mathbf{a}_i$$

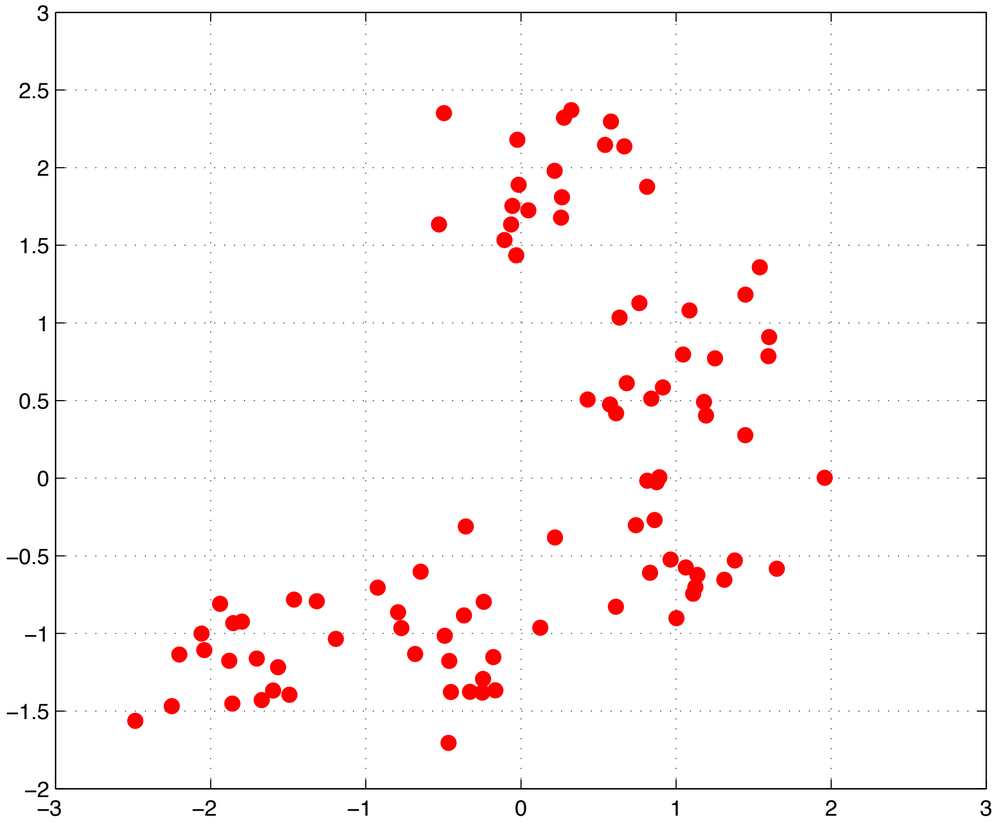
  - The non-linear principal components are a_i
  - The matrix $\mathbf{K}$ is now $N \times N$
- In practice we need to ensure that $\mathbf{K}$ is also positive definite (i.e. $\varphi(\mathbf{x})$ is zero mean)
  - Not hard to do, but we skip the details for now

An example

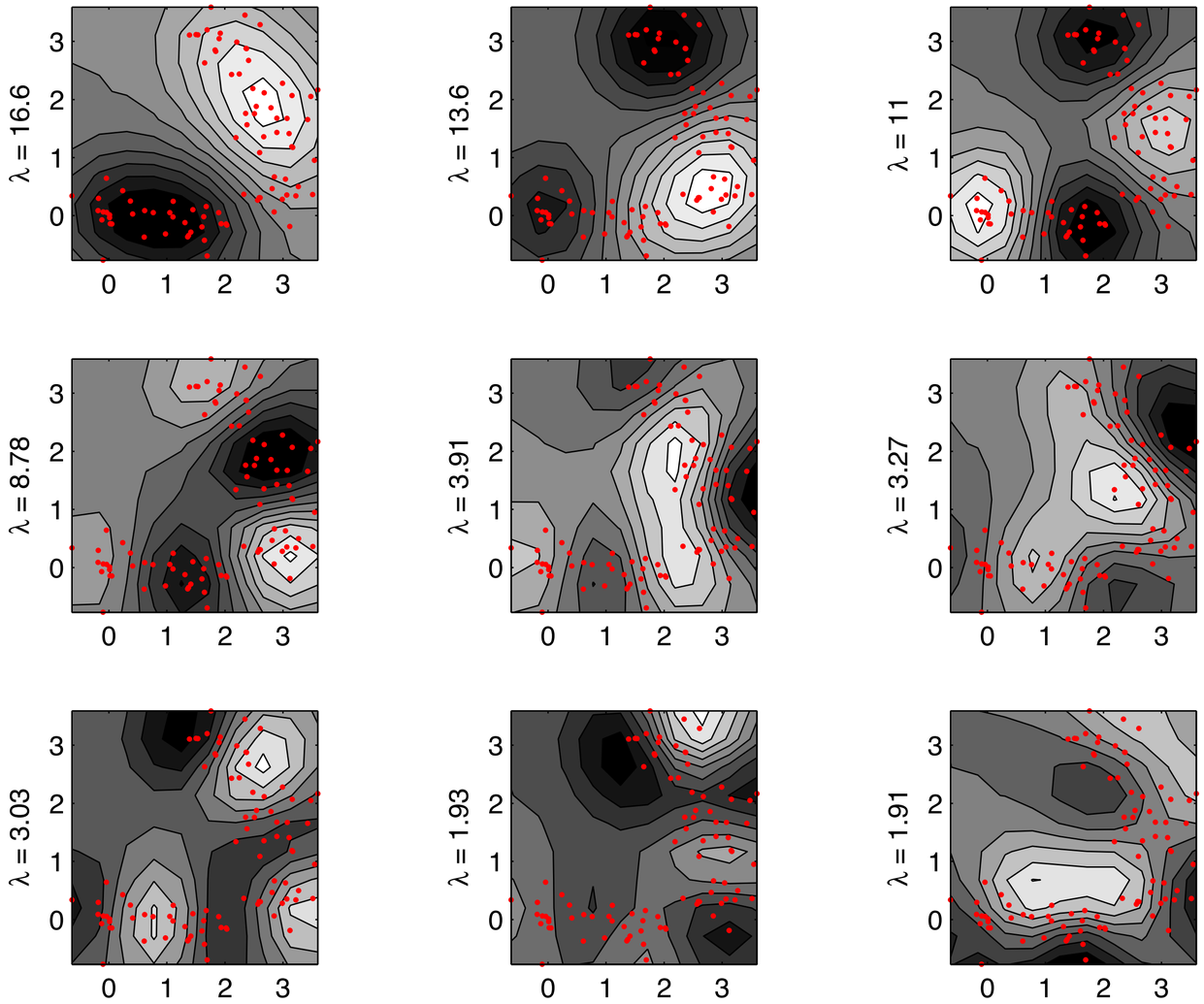
- Data is not good for PCA
  - too curvy, not Gaussian
- Use this kernel:

$$K(\mathbf{x},\mathbf{y}) = e^{-\left\|\mathbf{x}-\mathbf{y}\right\|^2}$$

  - Eigenvectors will be in terms of some high-D space



The non-linear eigenvectors



Some kPCA notes

- The eigenanalysis is now bigger
  - $\mathbf{K}$ is $N \times N$
    - Instead of $\mathbf{X} \cdot \mathbf{X}^\top$ we analyze $\varphi(\mathbf{X}^\top \cdot \mathbf{X})$
  - So computations will be slower
- How many components can we extract?
  - Not more than the original dimensions
    - There will be zero eigenvalues past that
- How do we choose the right kernel? Who knows …

A different take on PCA

- What happens when we only have quantified relationships between data?
  - E.g. distance or similarity metrics
- Can we do dimensionality reduction?
- How do we find the geometry of our data?
  - Very useful in psychology and social sciences

Reconstructing from distances

- Suppose we only have cross-data distances
  - Euclidean ones for now

- E.g.:

Distance matrix

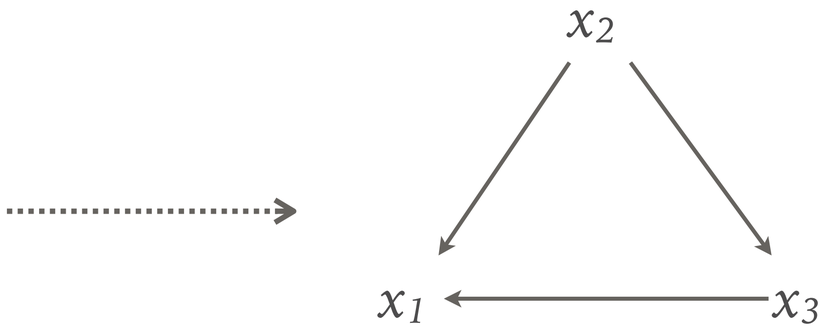
	x_1	x_2	x_3
x_1	0	1	1
x_2	1	0	1
x_3	1	1	0

Reconstructing from distances

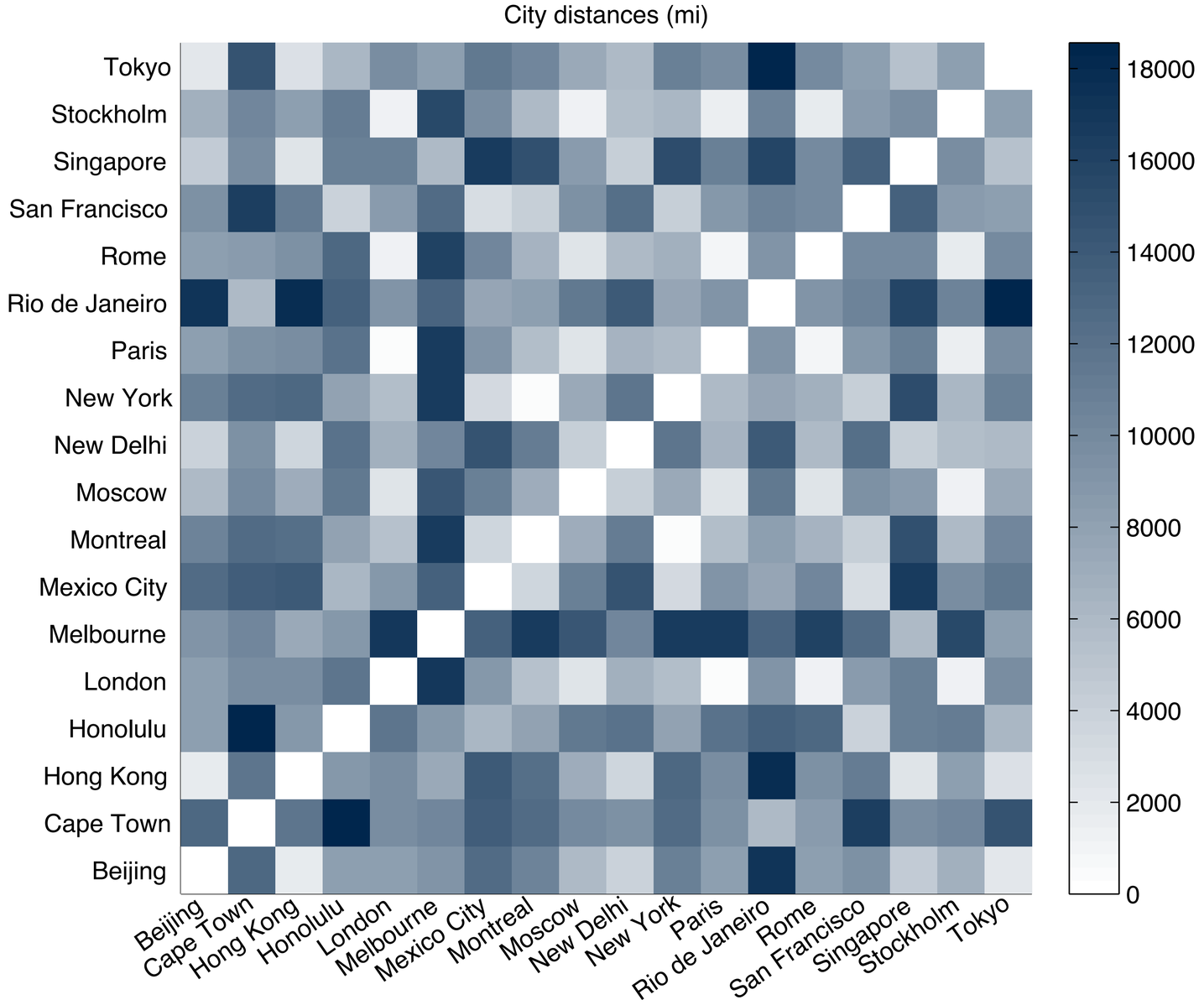
- Suppose we only have cross-data distances
  - Euclidean ones for now
- E.g.:

Distance matrix

	x_1	x_2	x_3
x_1	0	1	1
x_2	1	0	1
x_3	1	1	0



What about now?



Getting from distances to points

- Given only the cross-point distances:

$$d_{i,j} = \left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2$$

- We want to estimate $\mathbf{x}$, such that:

$$\left\| \hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j \right\|^2 \approx d_{i,j}$$

  - How do we go about it?

Writing this as a matrix operation

- We can express the squared distances as a product:

$$d_{i,j} = \left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2 = \mathbf{x}_i^\top \cdot \mathbf{x}_i + \mathbf{x}_j^\top \cdot \mathbf{x}_j - 2\mathbf{x}_i^\top \cdot \mathbf{x}_j \Rightarrow$$

$$\mathbf{D} = \underbrace{\operatorname{diag}\left(\mathbf{X}^\top \cdot \mathbf{X}\right) \cdot \mathbf{1}_N^\top}_{x_i^\top \cdot x_i \text{ replicated across columns}} + \underbrace{\mathbf{1}_N \cdot \operatorname{diag}\left(\mathbf{X}^\top \cdot \mathbf{X}\right)^\top}_{x_j^\top \cdot x_j \text{ replicated across rows}} - 2\mathbf{X}^\top \cdot \mathbf{X}$$

- If the first two terms were zero, we could solve:

$$\mathbf{D} = \mathbf{X}^\top \cdot \mathbf{X}$$

  - But they are not, so how do we remove the unwanted terms?
    - Let's average them out

Removing the unwanted terms

- To remove the row/column mean of a matrix do:

$$\mathbf{P} = \mathbf{I}_N - \frac{1}{N}\mathbf{1}_N \cdot \mathbf{1}_N^\top = \mathbf{I}_N - \frac{\mathbf{1}_{N\times N}}{N}$$

$$\underbrace{\left(\mathbf{A}\cdot\mathbf{P}\right)}_{\text{Remove mean column}} \cdot \frac{1}{N}\mathbf{1}_N = \mathbf{0}_N \qquad \frac{1}{N}\mathbf{1}_N^\top \cdot \underbrace{\left(\mathbf{P}\cdot\mathbf{A}\right)}_{\text{Remove mean row}} = \mathbf{0}_N^\top$$

- Applying on $\mathbf{D}$:

$$\mathbf{P}\cdot\mathbf{D}\cdot\mathbf{P} = \mathbf{P}\cdot\text{diag}\left(\mathbf{X}^\top\cdot\mathbf{X}\right)\cdot\mathbf{1}_N^\top\cdot\mathbf{P} + \mathbf{P}\cdot\mathbf{1}_N\cdot\text{diag}\left(\mathbf{X}^\top\cdot\mathbf{X}\right)\cdot\mathbf{P} - 2\mathbf{P}\cdot\mathbf{X}^\top\cdot\mathbf{X}\cdot\mathbf{P} =$$

$$= -2\mathbf{P}\cdot\mathbf{X}^\top\cdot\mathbf{X}\cdot\mathbf{P} = -2\left(\mathbf{X}\cdot\mathbf{P}\right)^\top\cdot\left(\mathbf{X}\cdot\mathbf{P}\right) = -2\left(\mathbf{X}-\bar{\mathbf{x}}\right)^\top\cdot\left(\mathbf{X}-\bar{\mathbf{x}}\right)$$

  - Bonus, a zero mean assumption on $\mathbf{x}$ (which we like!)

Overall process

- Get distances and "double center":

$$\mathbf{S} = -\frac{1}{2}\left(\mathbf{D} - \frac{1}{N}\mathbf{D}\cdot\mathbf{1}_N\cdot\mathbf{1}_N^\top - \frac{1}{N}\mathbf{1}_N\cdot\mathbf{1}_N^\top\cdot\mathbf{D} + \frac{1}{N^2}\mathbf{1}_N\cdot\mathbf{1}_N^\top\cdot\mathbf{D}\cdot\mathbf{1}_N\cdot\mathbf{1}_N^\top\right)$$

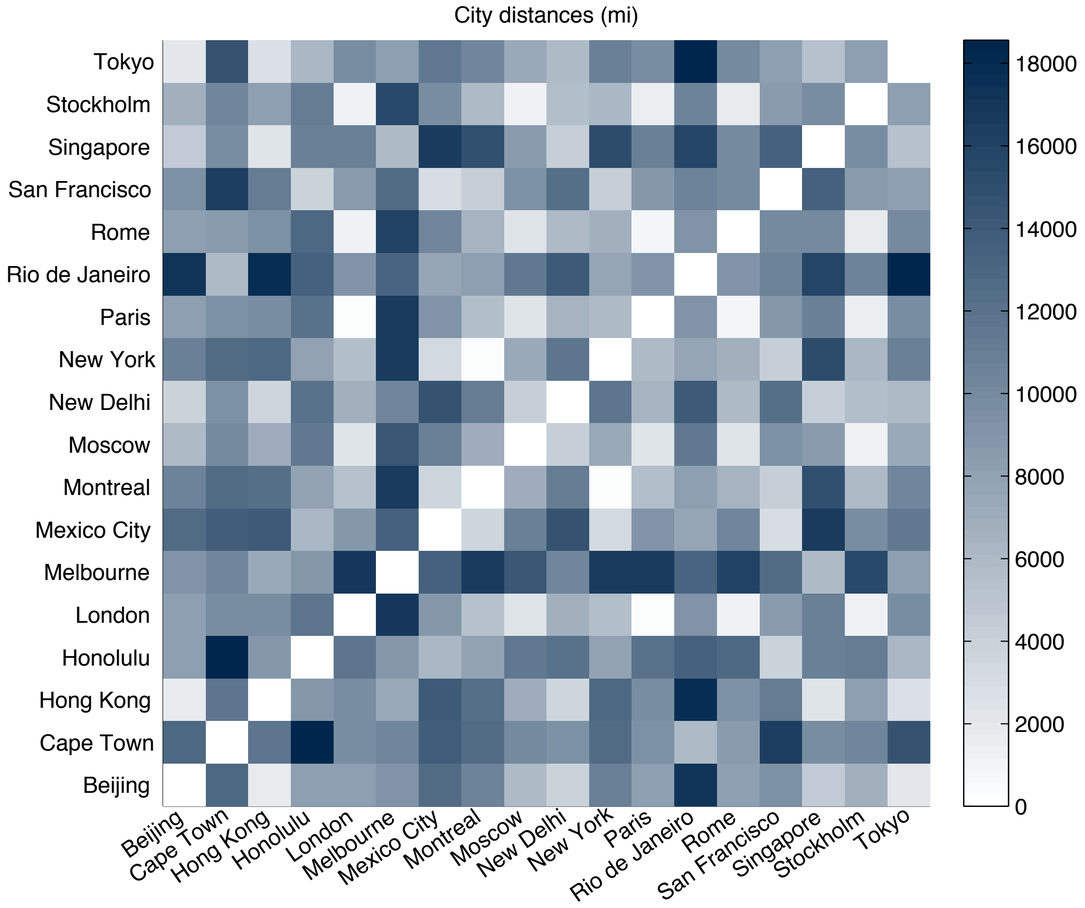
- Estimate $\mathbf{X}$ from $\mathbf{S}$

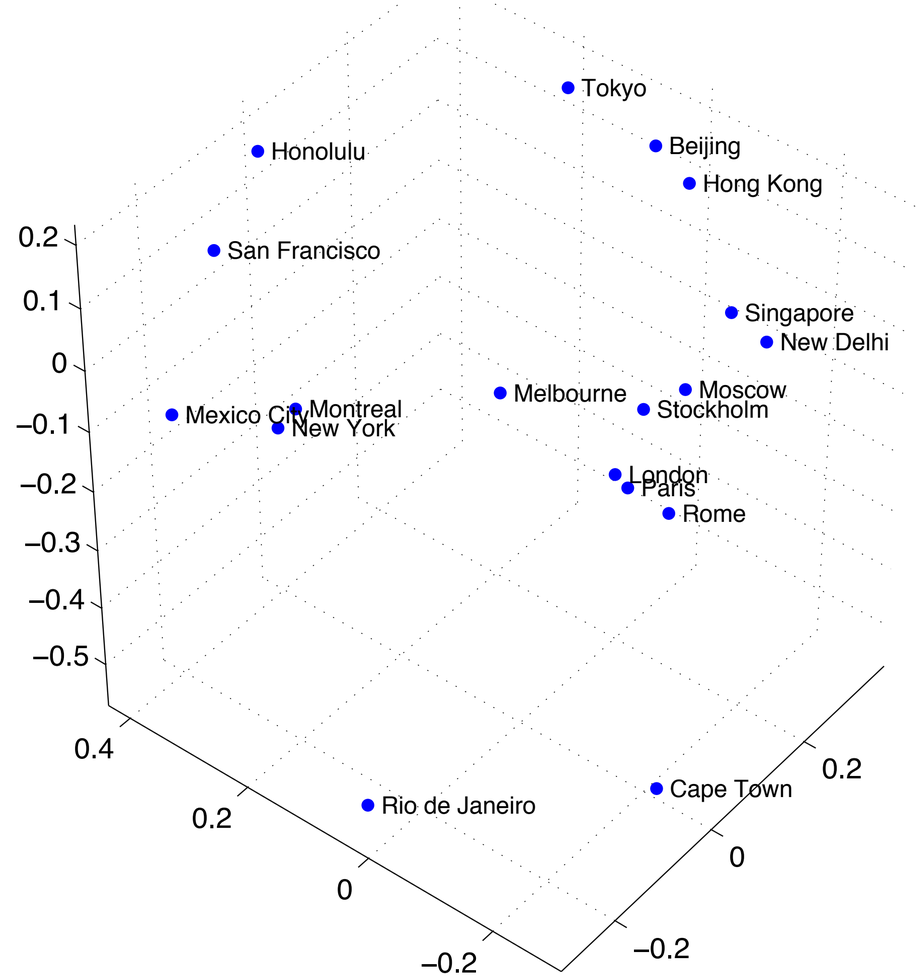
$$\mathbf{S} = \mathbf{X}^\top\cdot\mathbf{X} \Rightarrow \mathbf{S} = \underbrace{\mathbf{U}\cdot\Lambda\cdot\mathbf{U}^\top}_{\textit{Eigendecomposition}} \Rightarrow \hat{\mathbf{X}} = \Lambda^{1/2}\cdot\mathbf{U}^\top$$

- $\mathbf{X}$ is an N-dimensional representation of the data points
  - Keep as many dimensions you want
    - They are ordered by importance from the eigendecomposition!

Back to the problem at hand

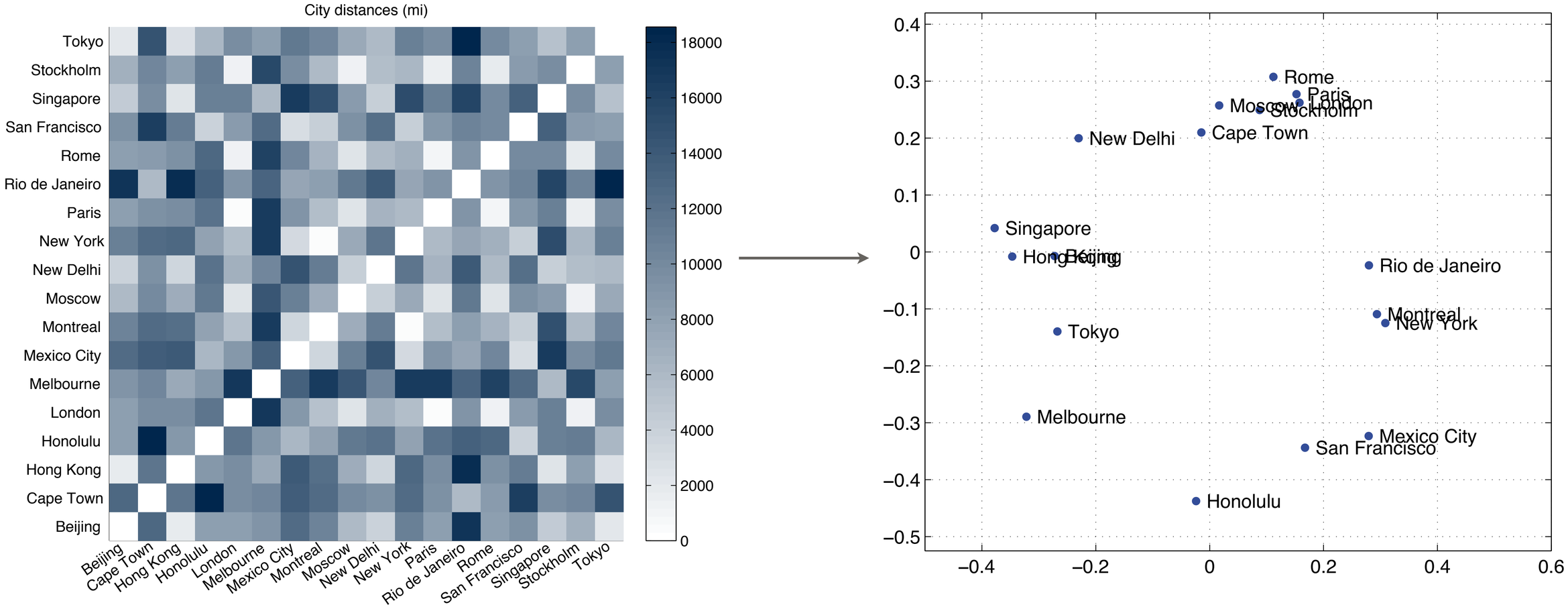
- City distance matrix results in an accurate map!





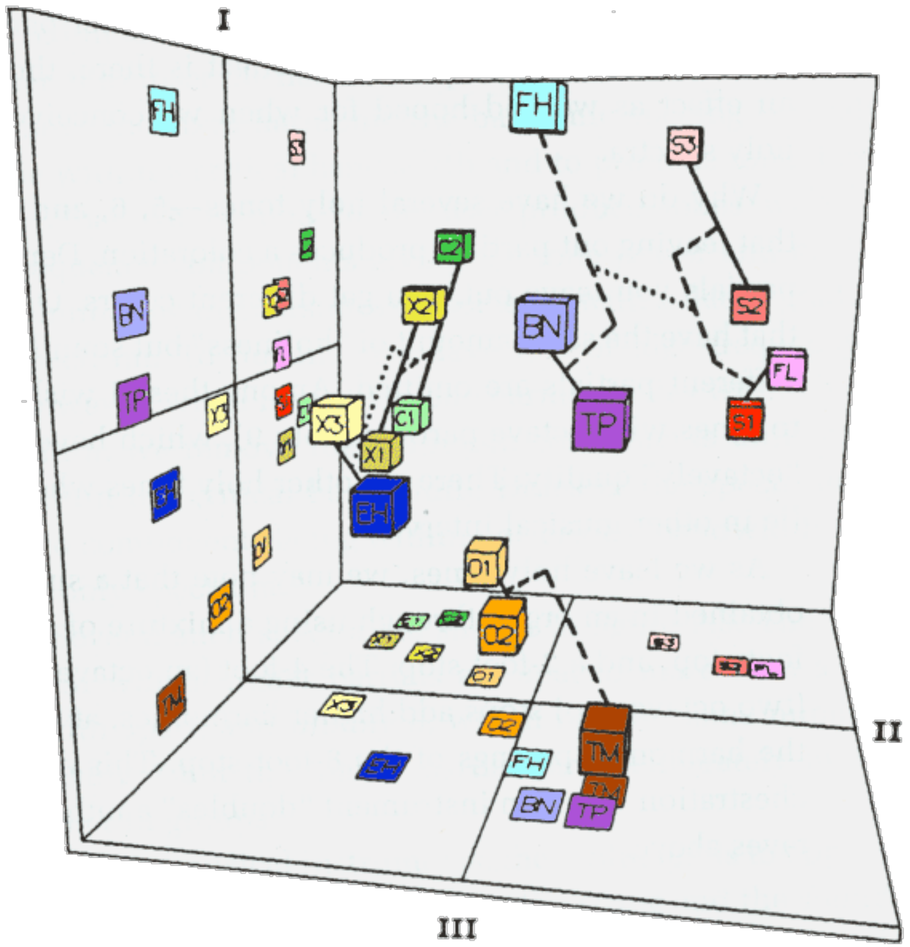
With fewer dimensions

- Also pretty accurate!



Another use

- Interpreting percepts
  - Get distances from human subjects
  - Create map of percept
- Gray's timbre space



Variations

- Preserving dot products
- Using ranking as opposed to distances
- Using non-linear distance metrics
- Using only local data

What is MDS good for

- Finds the implied geometry of our data
  - Not obvious for large dimensions!

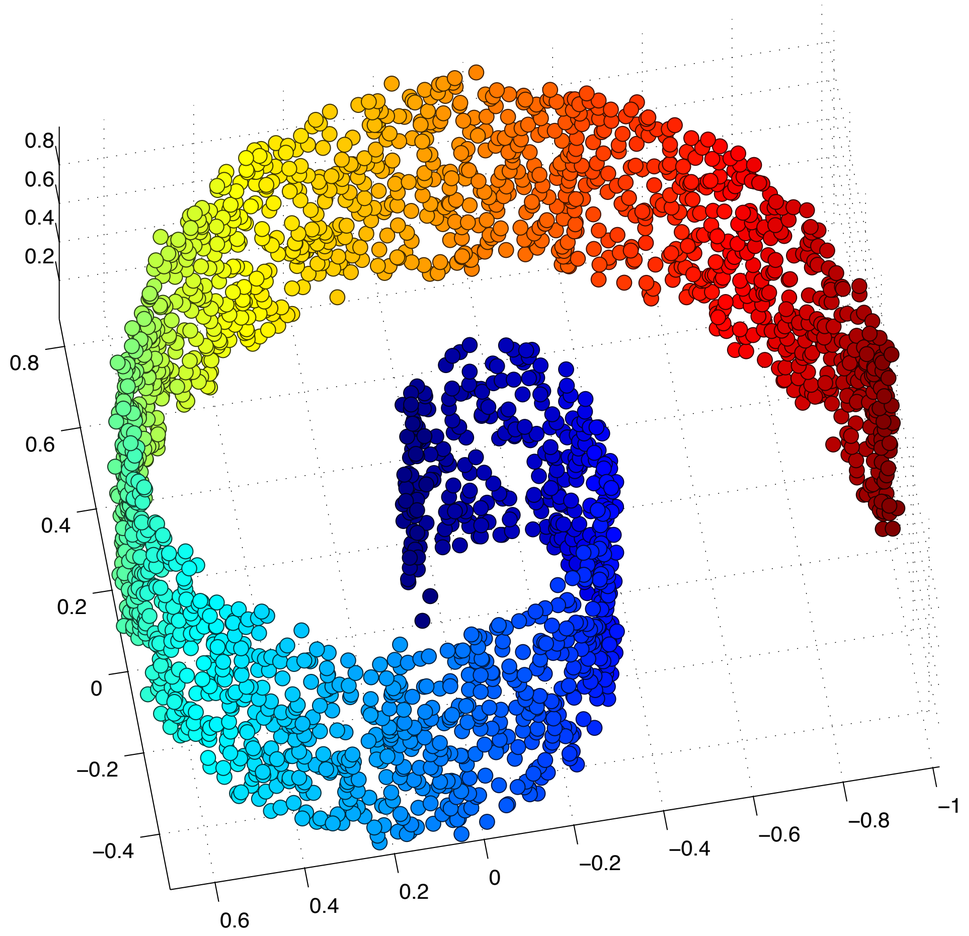
- It is a sort of like PCA
  - Like PCA: But decomposes $\mathbf{X}^\top \cdot \mathbf{X}$ not $\mathbf{X} \cdot \mathbf{X}^\top$
  - Like kPCA: The kernel is the inner product
  - Is linear

Data with low intrinsic dimension

- Actual dimension can be misleading

Data with low intrinsic dimension

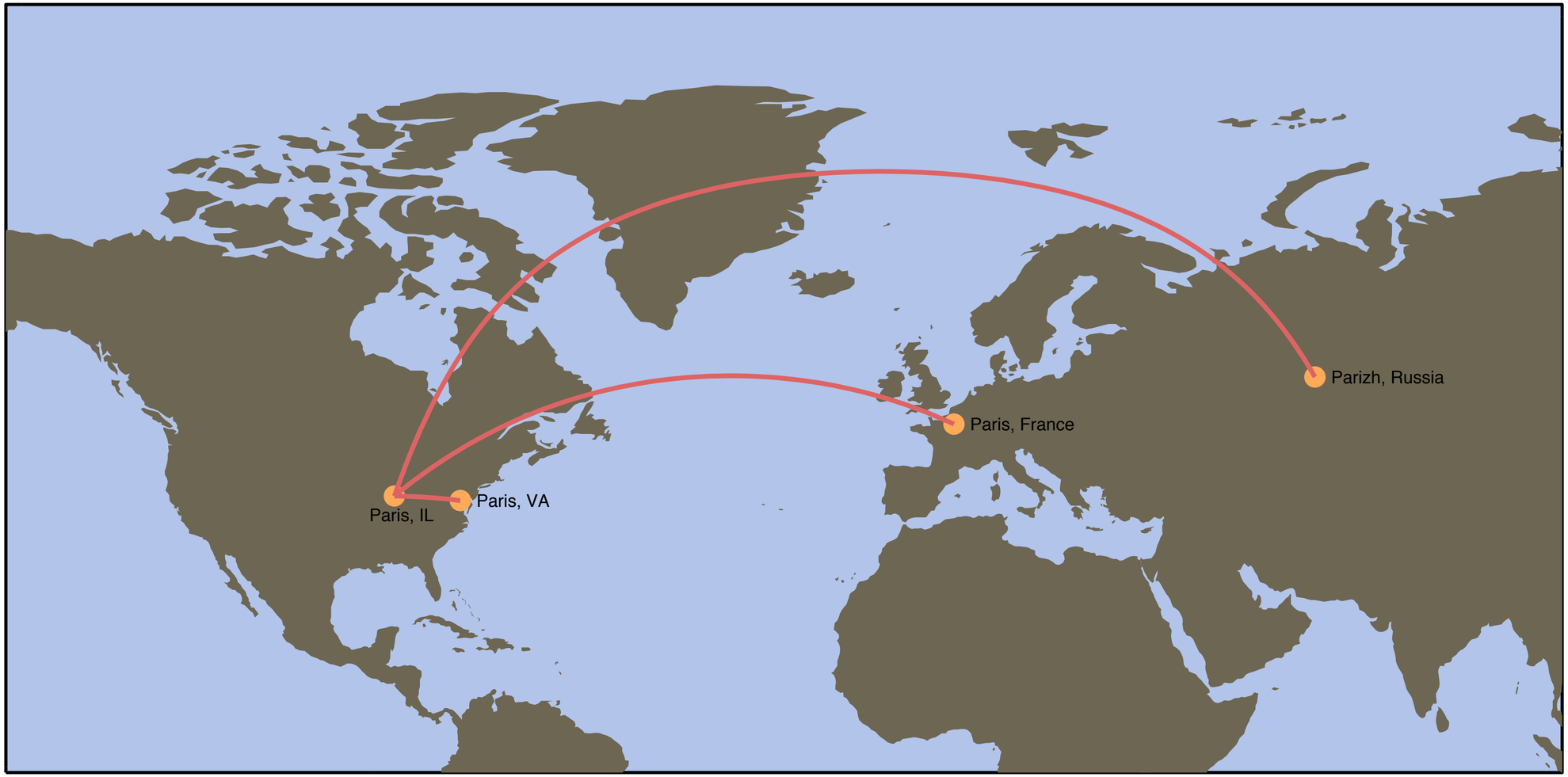
- Actual dimension can be misleading



Local vs. Global

Local vs. Global

Local vs. Global



Preserving topology

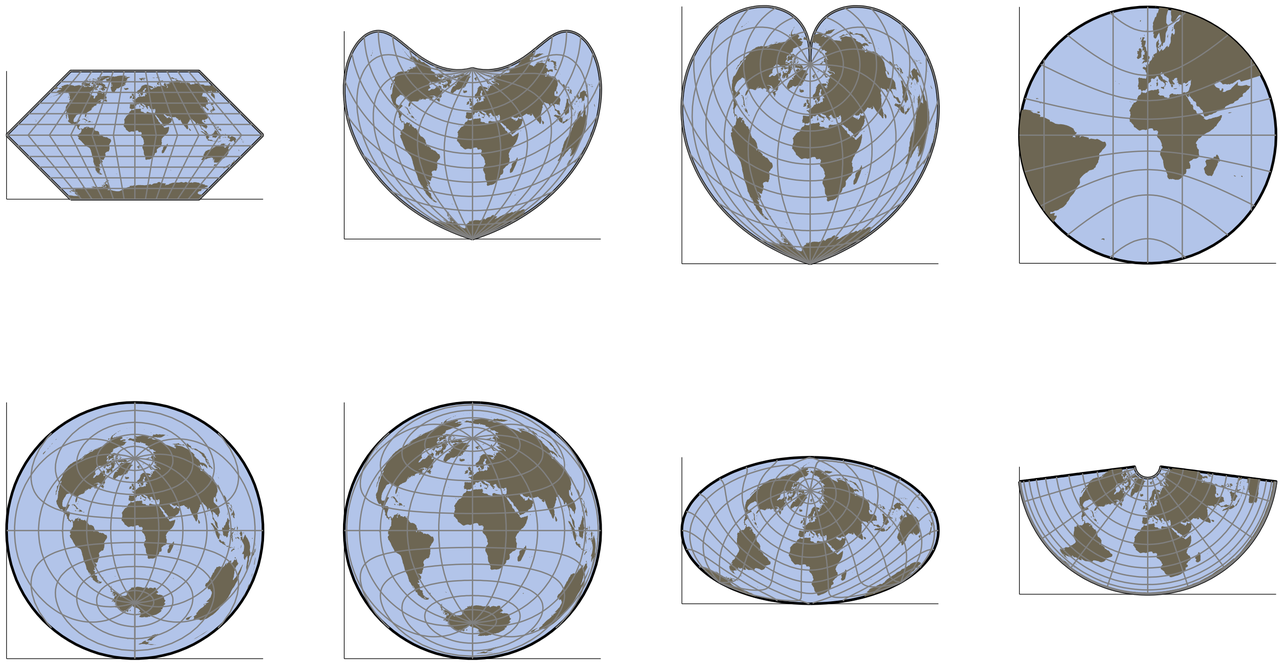
- Looking at the neighboring structure
  - Is the earth flat or round?

- Reduce dimensionality while keeping the neighborhood structure unchanged

Example: Map projections

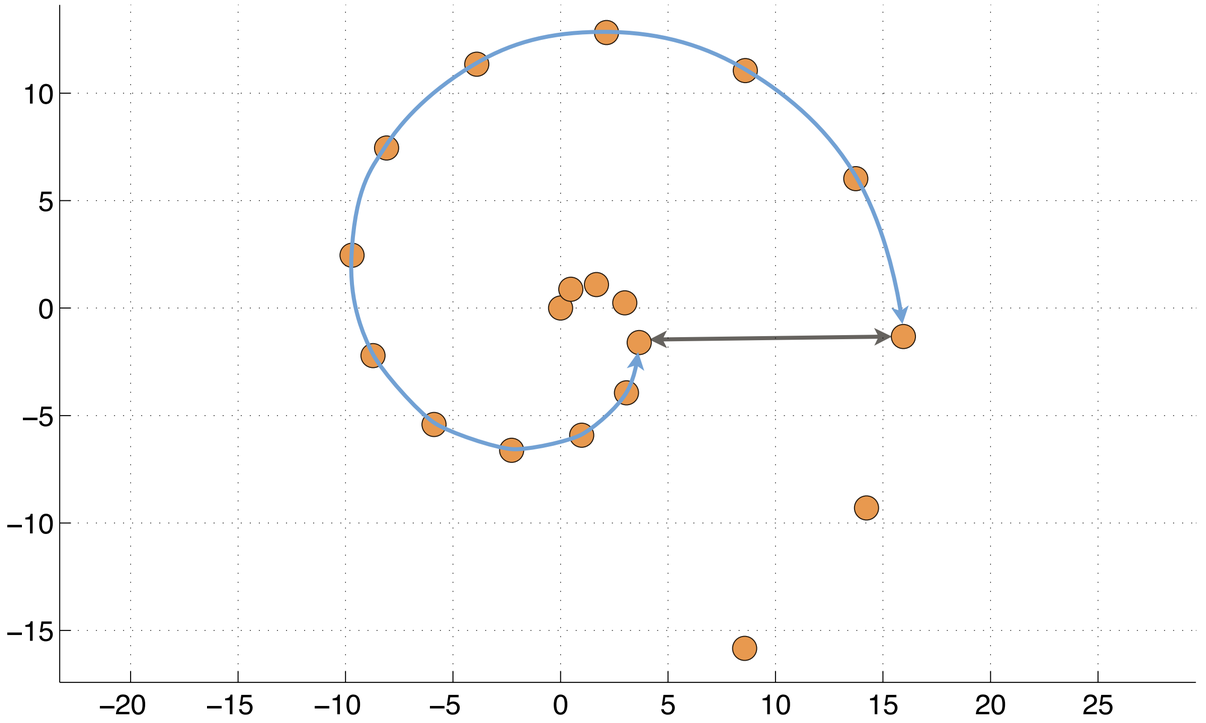
- Many ways to go from 3D to 2D
  - Depends on what you care about
    - How do we do this for data?

Dimensionality reduction



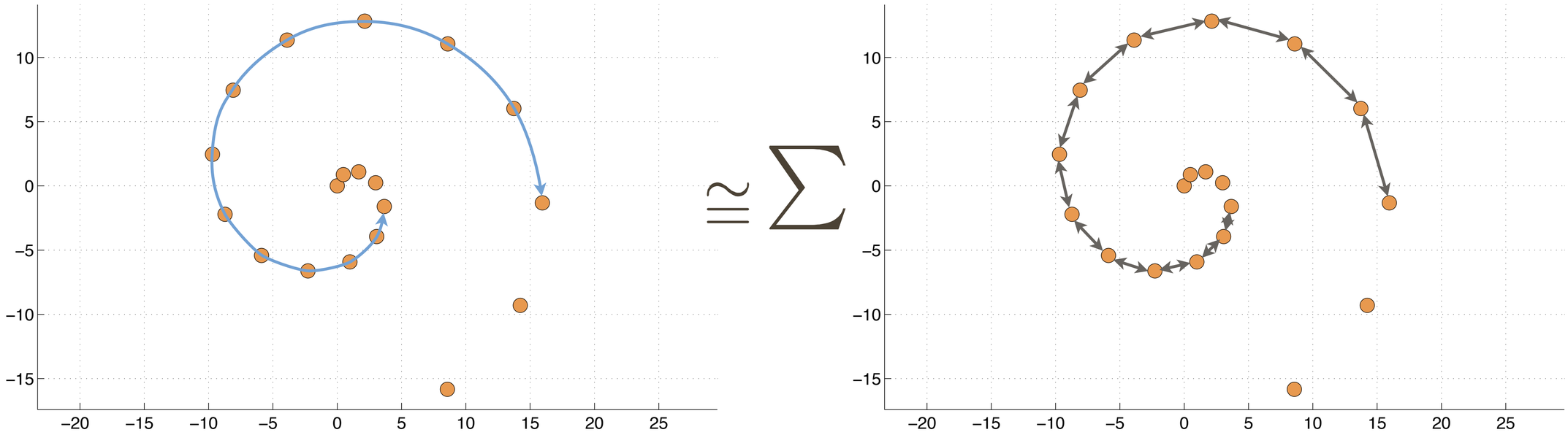
Euclidean vs. Geodesic distance

- On a manifold distance is subjective
  - Euclidean distance ignores manifold structure
  - Geodesic distance is more appropriate



Getting the distance using just data

- Since we don't know the manifold we can use the available data to get the geodesic distance
  - Shortest path between two points that passes through neighboring data

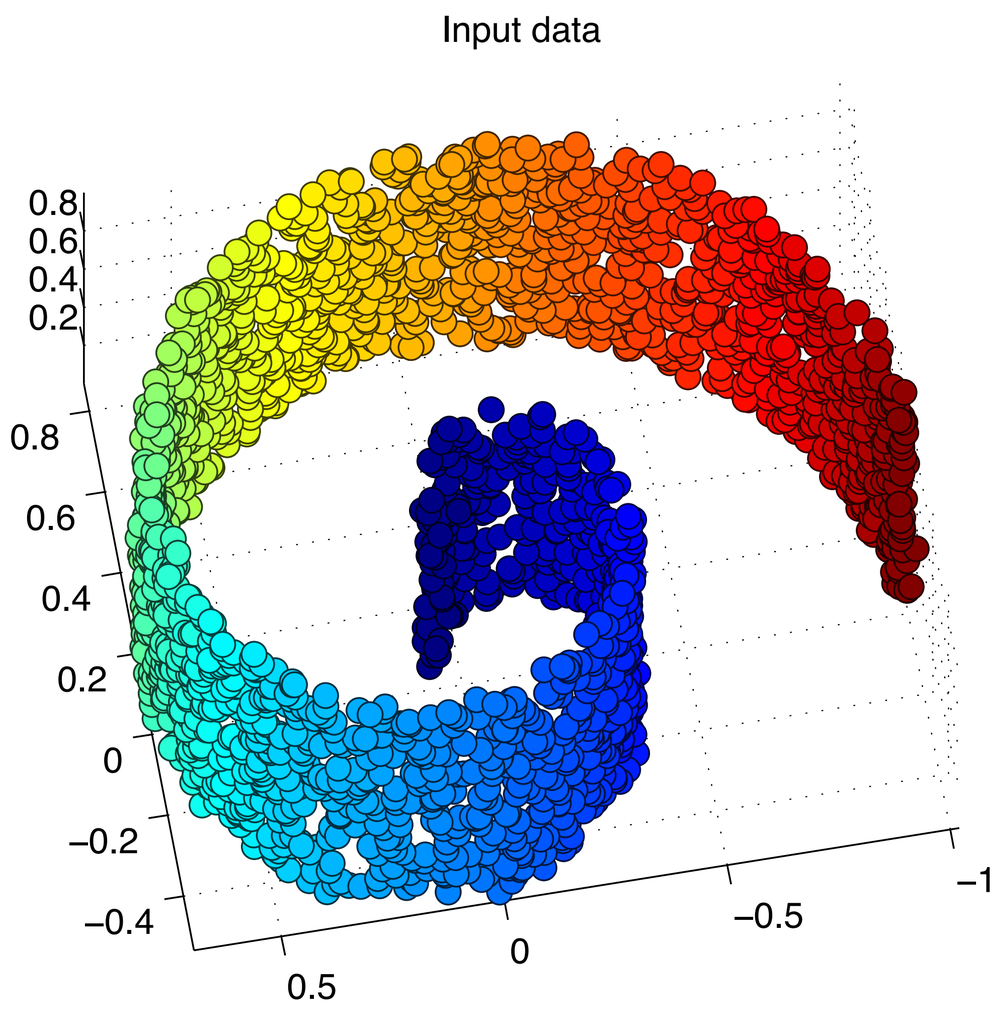


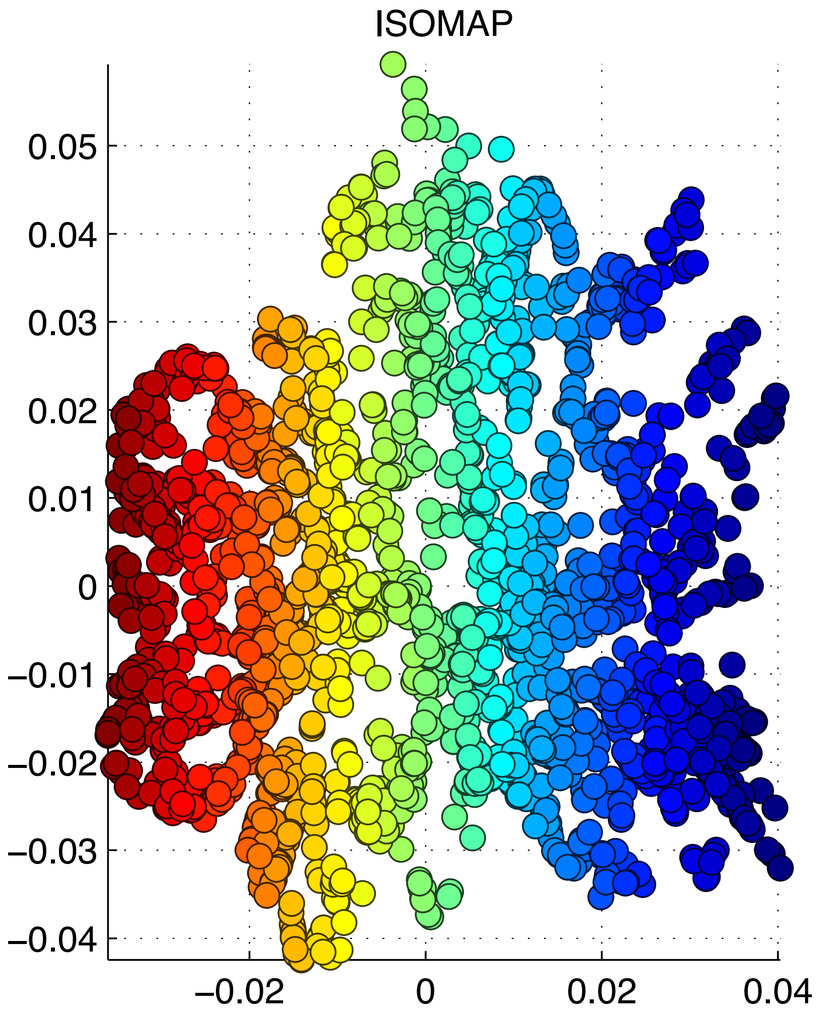
ISOMAP

- Perform MDS using geodesic distances
  - i.e. change only $\mathbf{D}$, the matrix with the distances

- Results in an embedding that's aware of the low-dimensional structure in our data
  - Results in Euclidean distances that are approximately equal to the geodesic distances

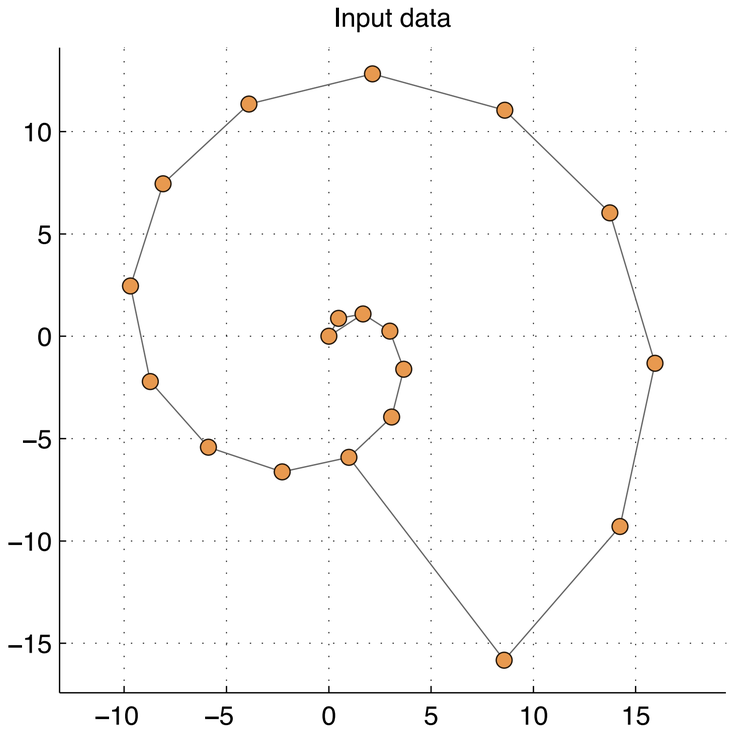
On the Swiss roll

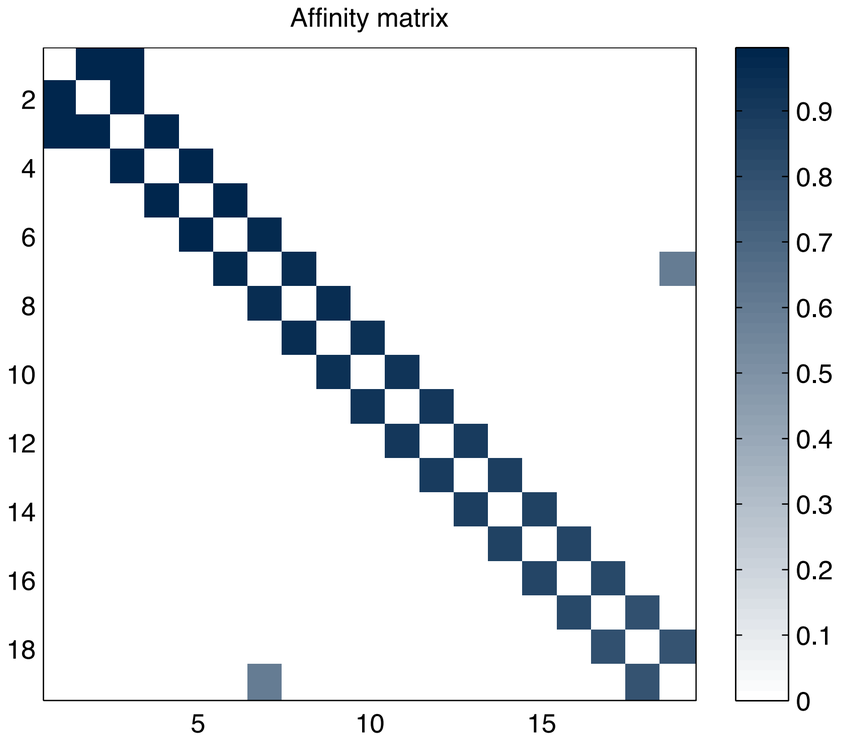




A neighborhood graph approach

- Make a note of point distances
  - N-nearby points matter, zero out the rest (and the diagonal)





$$w_{i,j} = \mathrm{e}^{-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{\sigma}}$$

A goal

- Get an embedding that minimizes:

$$E_{LE} = \sum_{i,j} \left\| \mathbf{y}_i - \mathbf{y}_j \right\|^2 w_{i,j}$$

	Large $\lvert \mathbf{y}_i - \mathbf{y}_j \rvert$	Small $\lvert \mathbf{y}_i - \mathbf{y}_j \rvert$
Large w_{ij}	***Bad***	*Good*
Small w_{ij}	*Good*	*Don't care*

- We can rewrite as:

$$E_{LE} = 2\mathbf{Y}^\top \cdot \mathbf{L} \cdot \mathbf{Y}$$

$$\mathbf{L} = \mathbf{W} - \text{diag}\left(\mathbf{1}^\top \cdot \mathbf{W}\right) = \mathbf{W} - \mathbf{D}$$

Laplacian of the distance graph

Sum of rows

And we solve

- Impose a constraint against arbitrary scaling:

$$\text{minimize } \mathbf{Y}^\top \cdot \mathbf{L} \cdot \mathbf{Y}$$

$$\text{subject to } \mathbf{Y}^\top \cdot \mathbf{D} \cdot \mathbf{Y} = \mathbf{I}$$

To avoid $y_i = 0$ and arbitrary scaling

- one more rewrite:

$$\text{minimize } \mathbf{Z}^\top \cdot \tilde{\mathbf{L}} \cdot \mathbf{Z}$$

$$\text{subject to } \mathbf{Z}^\top \cdot \mathbf{Z} = \mathbf{I}$$

$$\mathbf{Z} = \mathbf{D}^{1/2} \cdot \mathbf{Y}$$

$$\tilde{\mathbf{L}} = \mathbf{D}^{-1/2} \cdot \mathbf{L} \cdot \mathbf{D}^{-1/2}$$

Normalized graph Laplacian

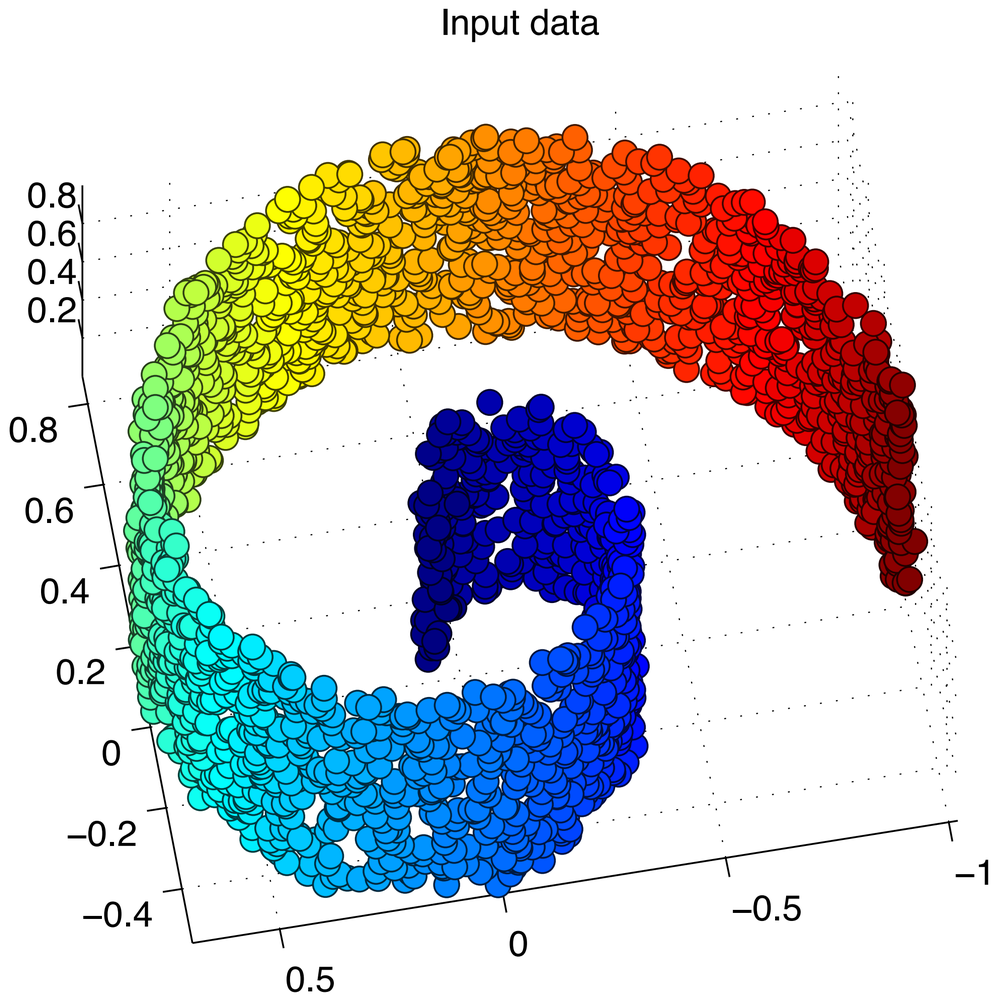
And predictably …

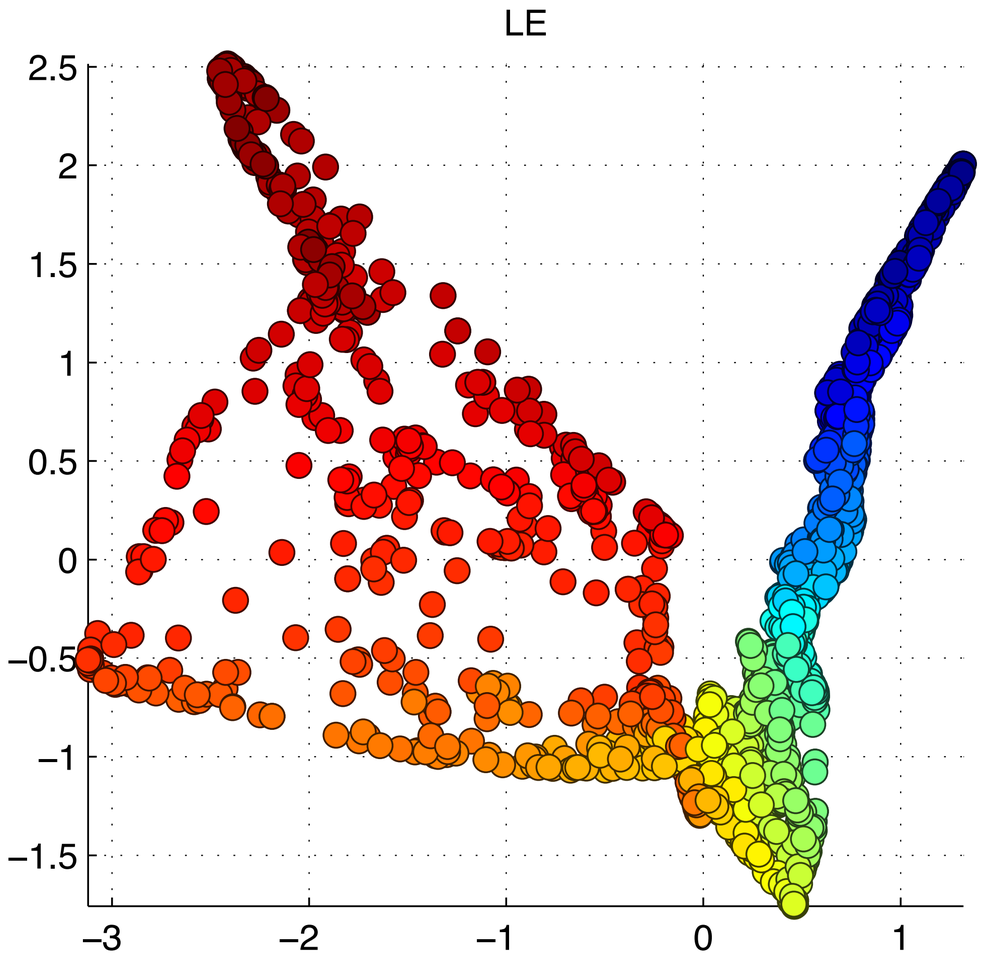
- $\mathbf{Z}$ are the eigenvectors of the normalized Laplacian
  - From which we get $\mathbf{Y}$

$$\mathbf{Y} = \mathbf{Z}^{\top} \cdot \mathbf{D}^{1/2}$$

- The eigenvalues will help sort $\mathbf{Y}$
  - Small eigenvalues imply closer spaced $\mathbf{y}$'s
    - The smallest eigenvalue will have all $\mathbf{y}$'s on one point (useless)
- So we pick the 2^{nd} to $N^{th}+1$ smallest eigenvectors to produce an embedding for N dimensions

Swiss roll example





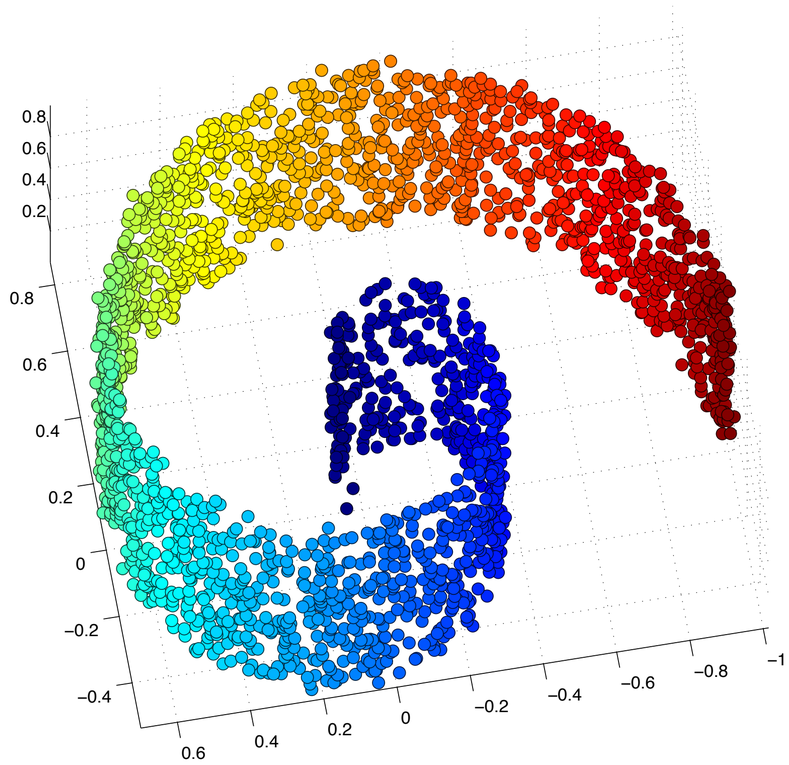
Locally Linear Embedding

- Observe local neighborhood of all points
  - Assume each neighborhood is linear
  - Explain each point using its neighbors

$$\mathbf{x}_i \approx \sum_{j \in N(i)} w_{i,j} \mathbf{x}_j$$

- And there's an optimal $\mathbf{W}$

$$C_H(\mathbf{W}) = \sum_i \left\| \mathbf{x}_i - \sum_{j \in N(i)} w_{i,j} \mathbf{x}_j \right\|^2$$

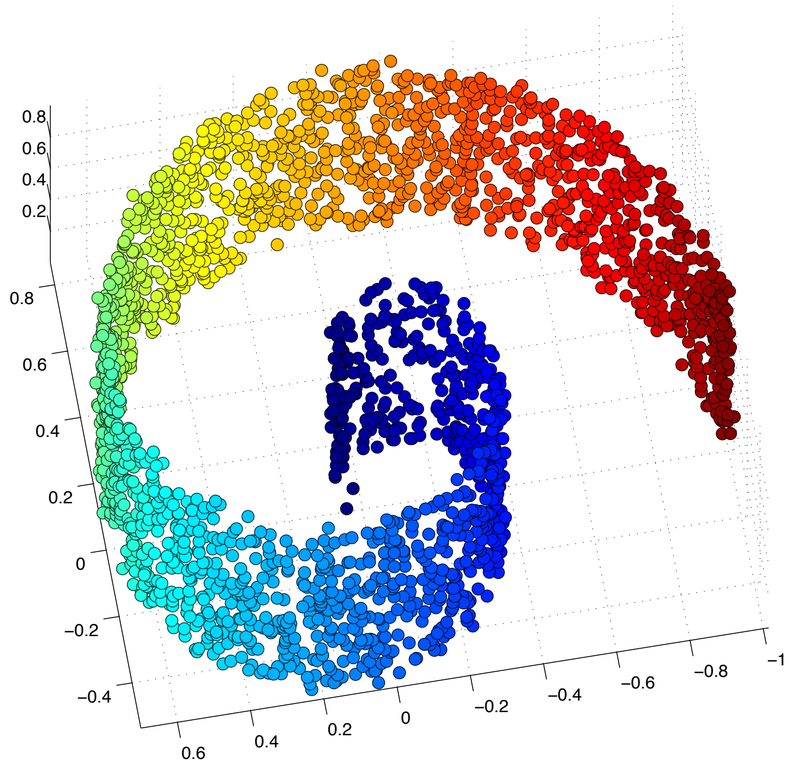


Locally Linear Embedding

- The weights will work just as well in a lower dimensional space
  - We assume local linearity

$$C_L(\mathbf{W}) = \sum_i \left\| \mathbf{y}_i - \sum_{j \in N(i)} w_{i,j} \mathbf{y}_j \right\|^2$$

- We now need to find a $\mathbf{y}$ that minimizes the above expression

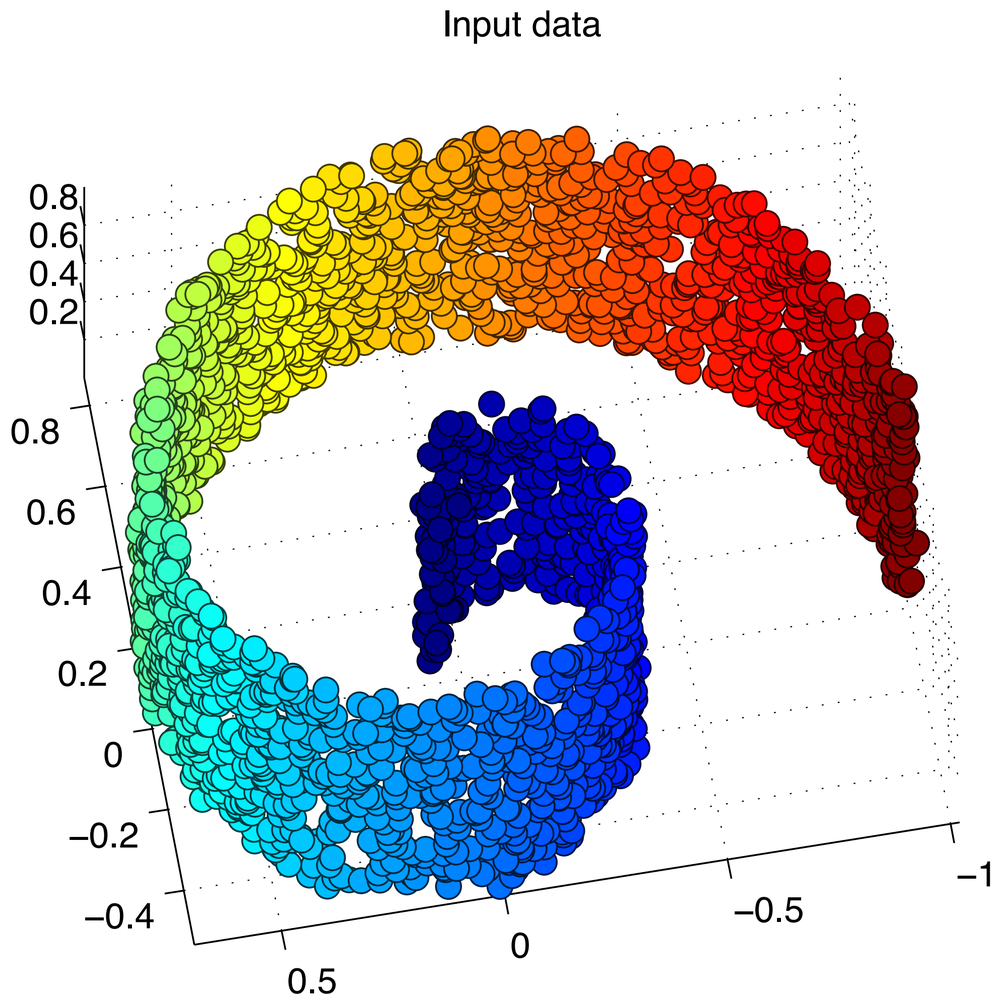


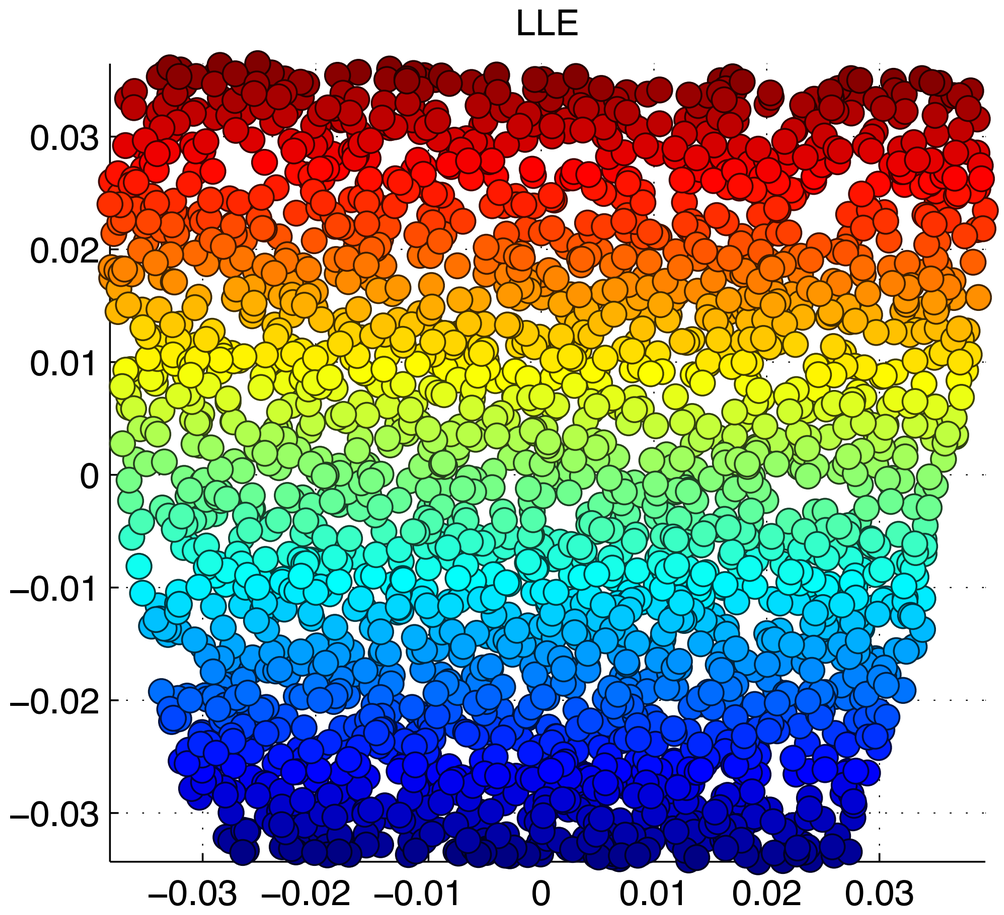
How to

- Impose constraint on $\mathbf{W}$
  - Rows must sum to 1
    - Make rotation, scale, translation invariant

- This becomes an eigendecomposition problem again!
  - This time we eigendecompose $(\mathbf{I}-\mathbf{W})^{\top}\cdot(\mathbf{I}-\mathbf{W})$
  - Discard smallest eigenvector (has zero eigenvalue)
  - The rest of the smallest eigenvectors will contain variates of $\mathbf{y}$

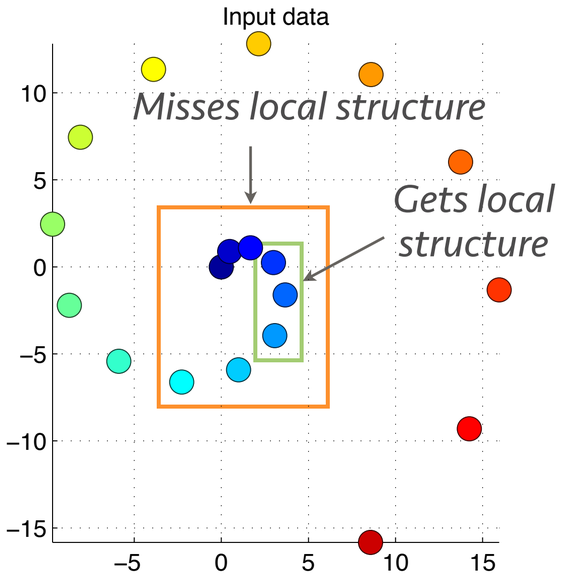
Swiss roll example

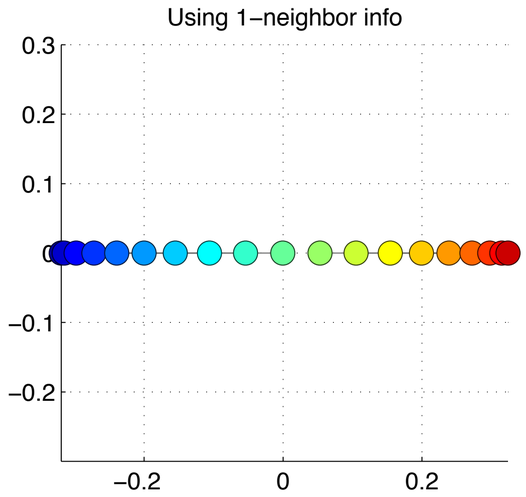




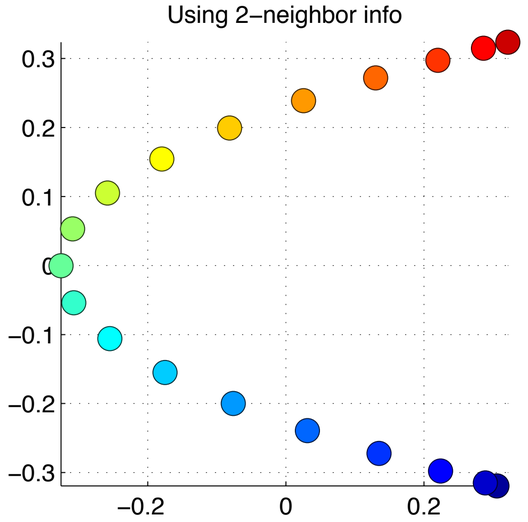
A note on manifold methods

- Try to highlight the local neighborhoods
  - e.g. clip the adjacency matrix, ignore distant points, etc.

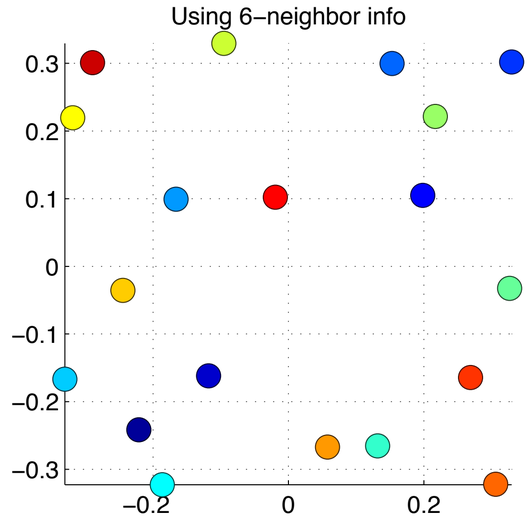




Sees the manifold as a line



Sees the manifold as a curve



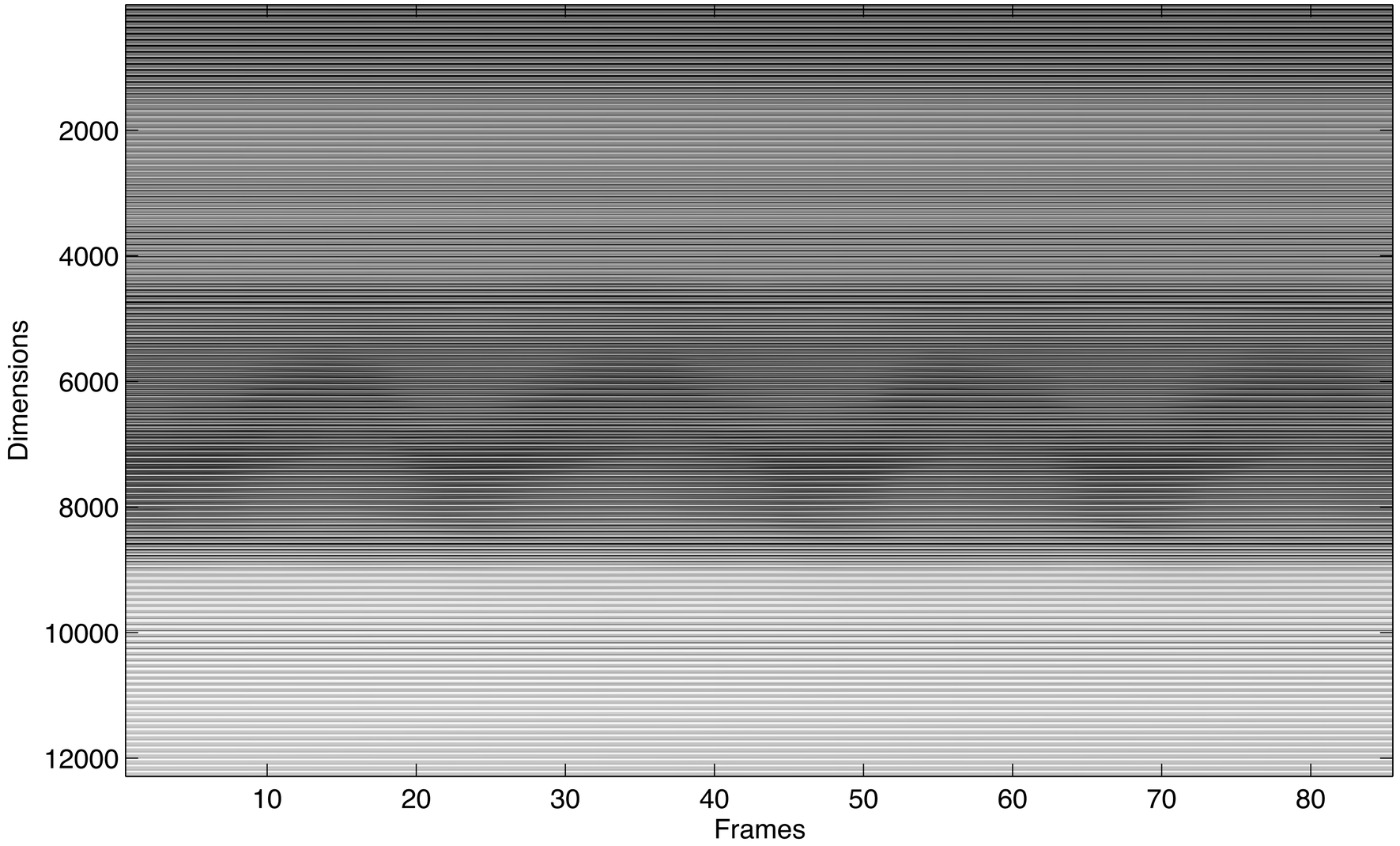
Sees the manifold as a mess!

A video example

- A high dimensional input
  - 240 x 320 pixels = 76,800 dimensions
- Low dimensional structure
  - Moving lips around
- Can we simplify the data?

Each frame is a pose

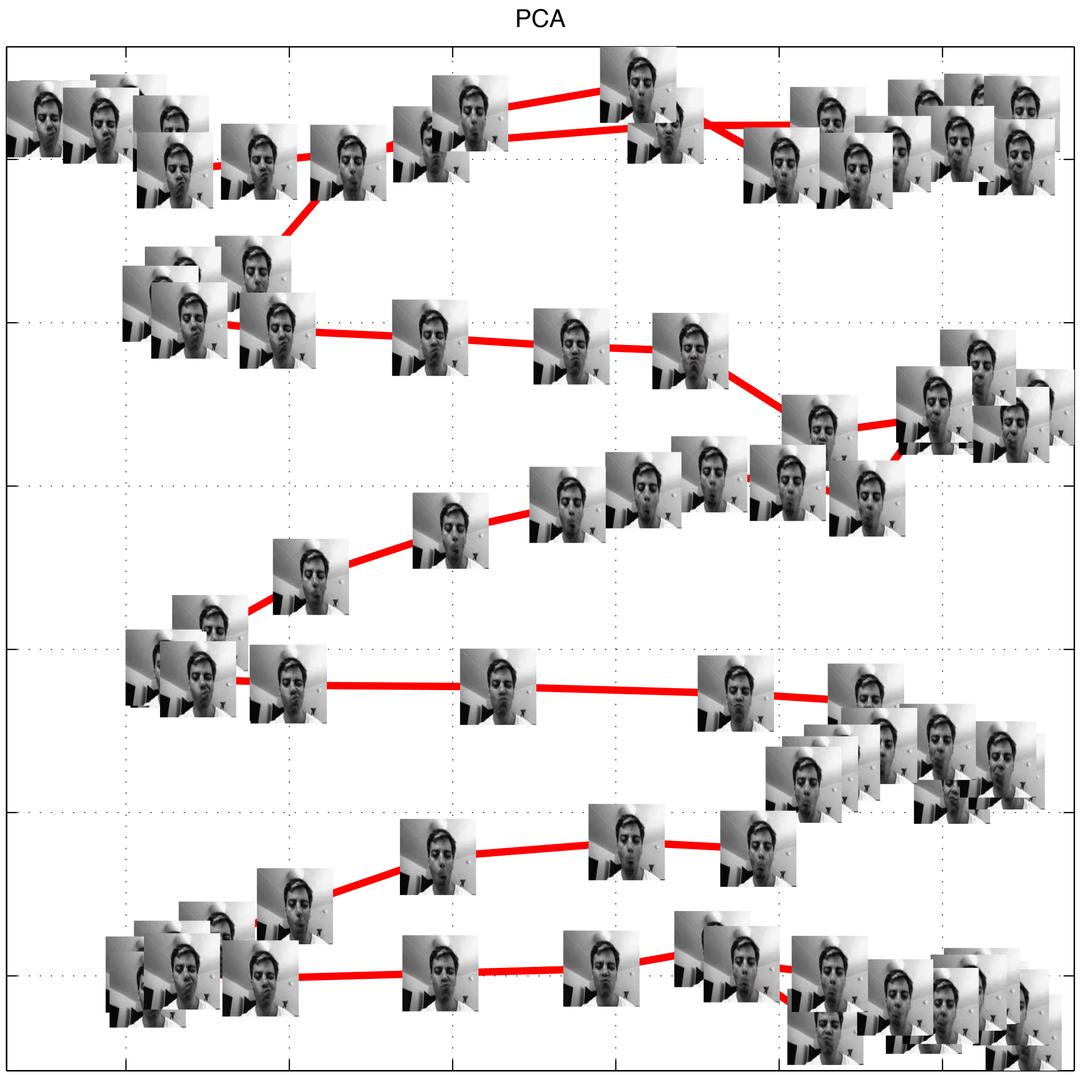
When unraveled



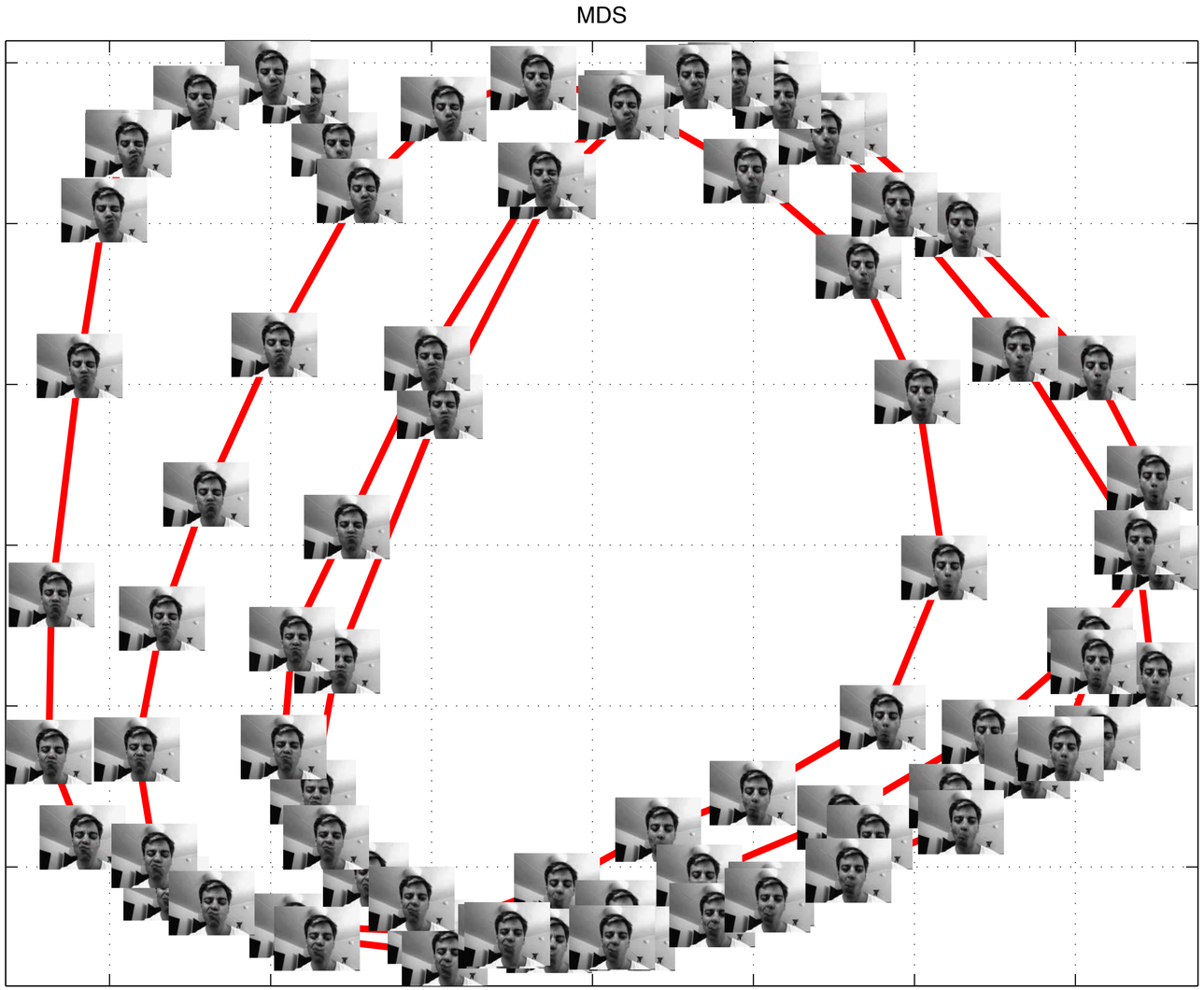
The manifold

- We only have a couple of dimensions
  - The primary being the movements of the lips
- The actual space is much lower dimensional than the video is
  - It is smooth
  - And it is “circular”

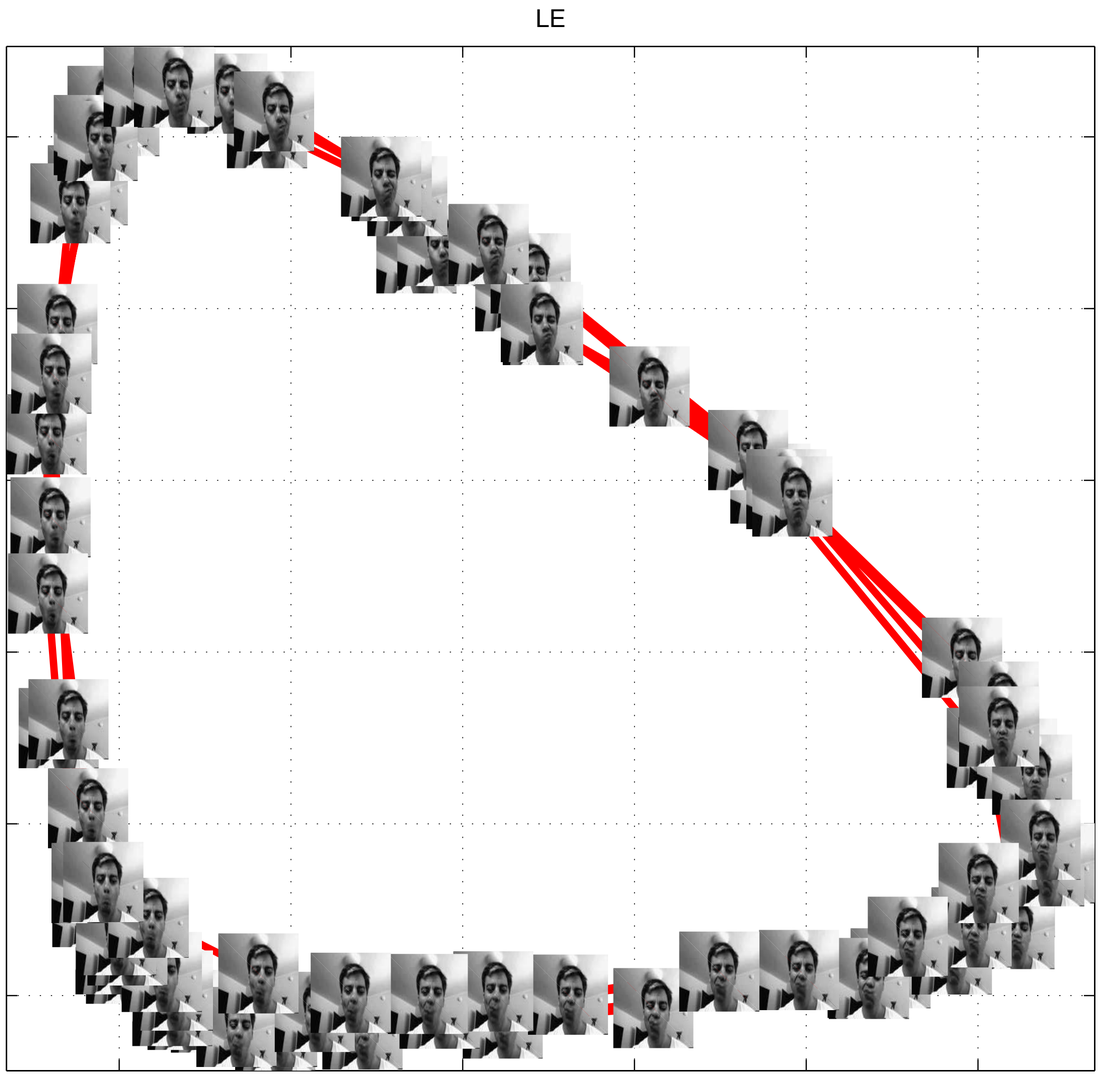
PCA



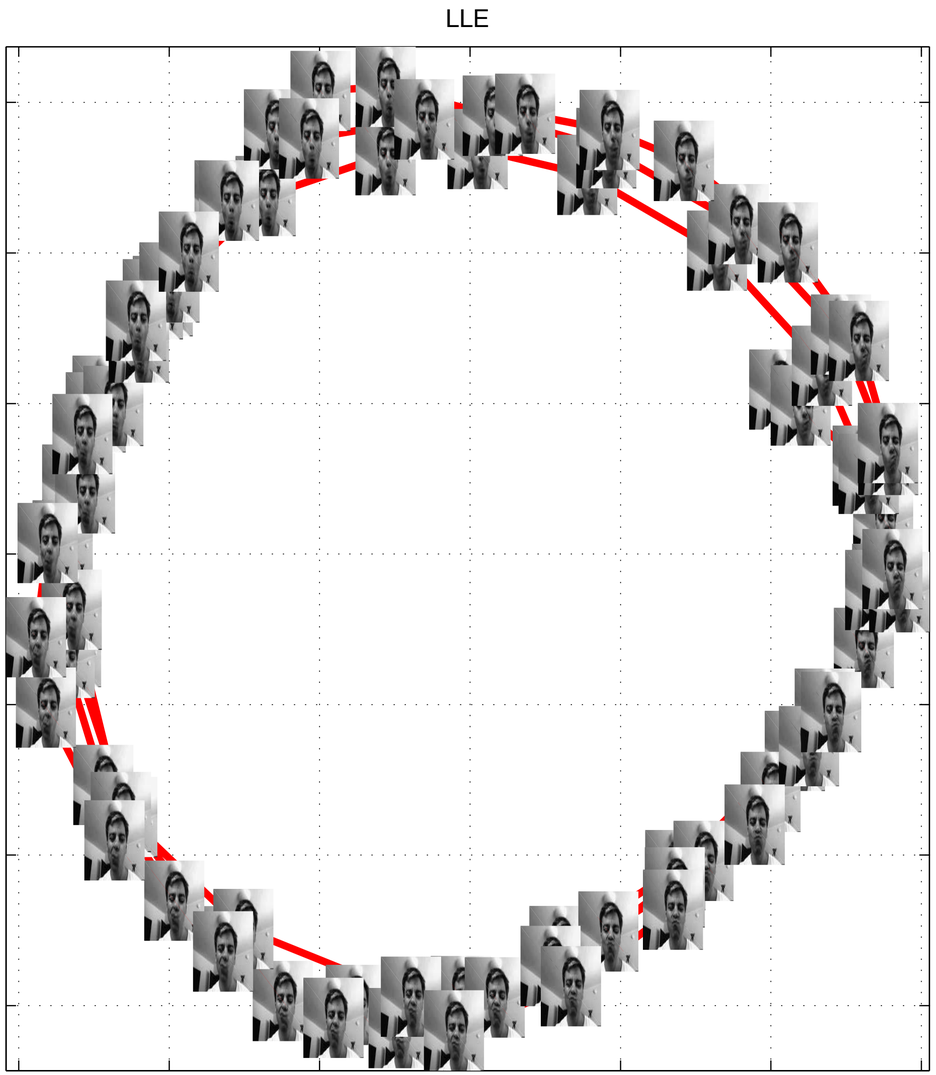
MDS



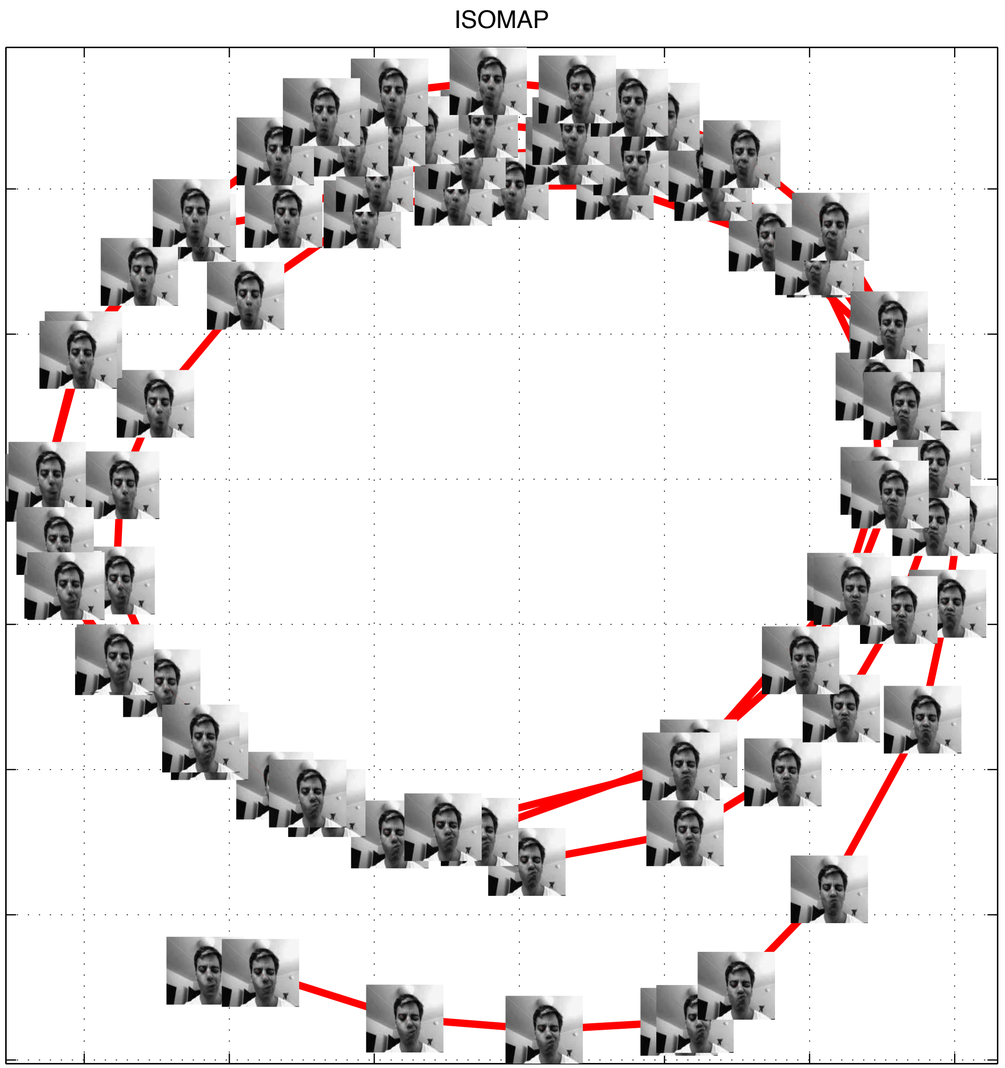
Laplacian Eigenmaps



LLE



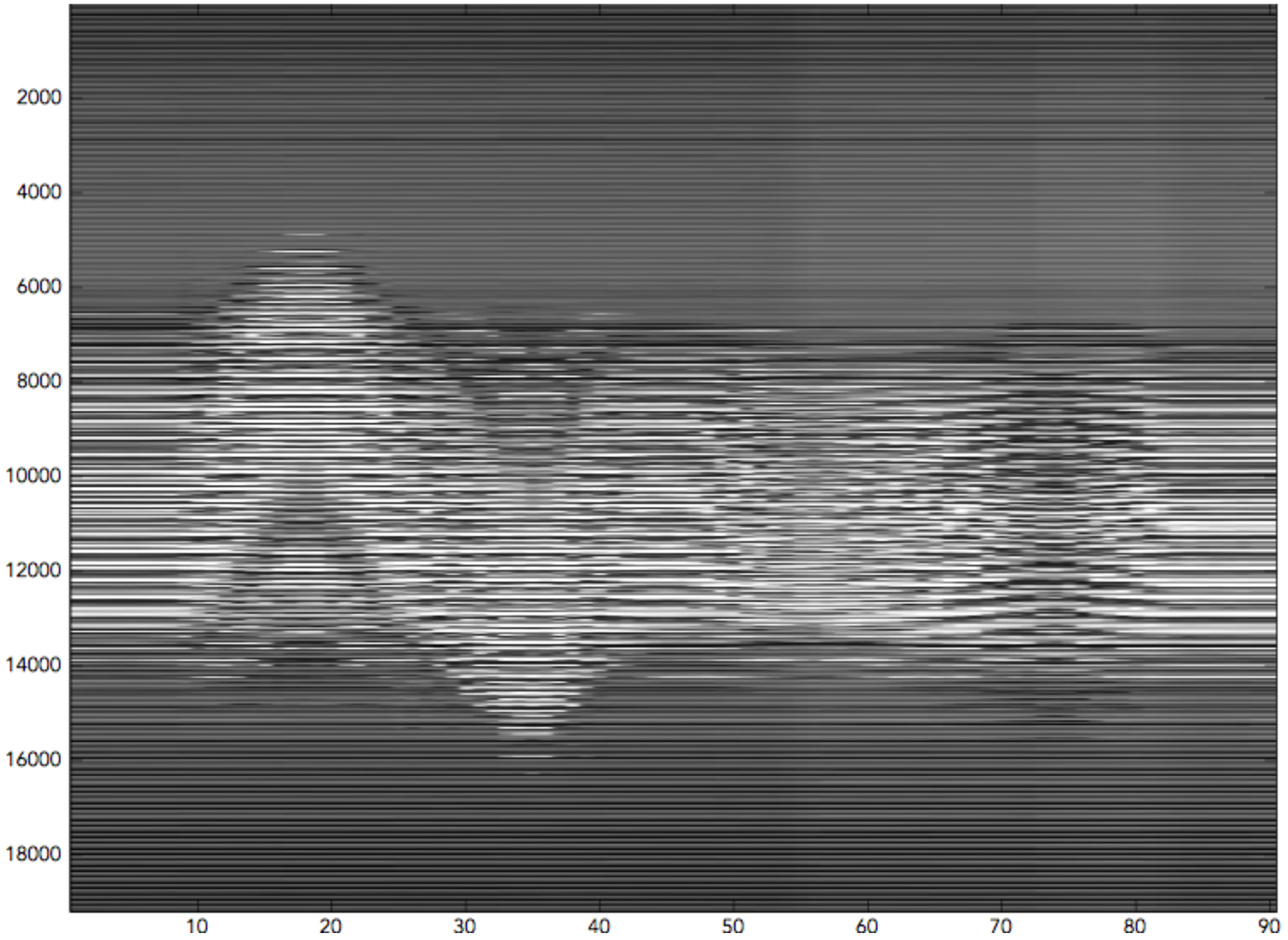
ISOMAP



Another example movie

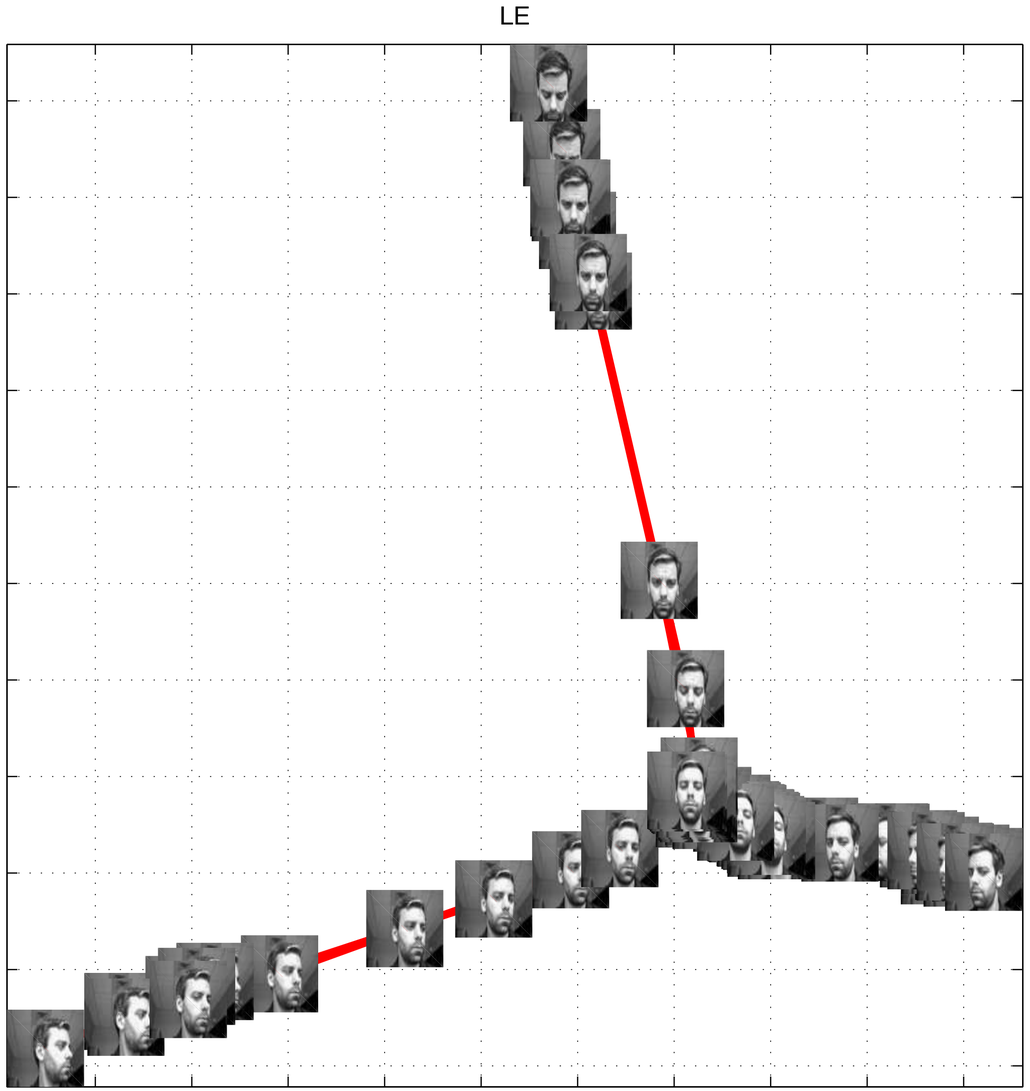
- This time we have distinct axes
  - Up-Down-Left-Right head movement

Input as a matrix



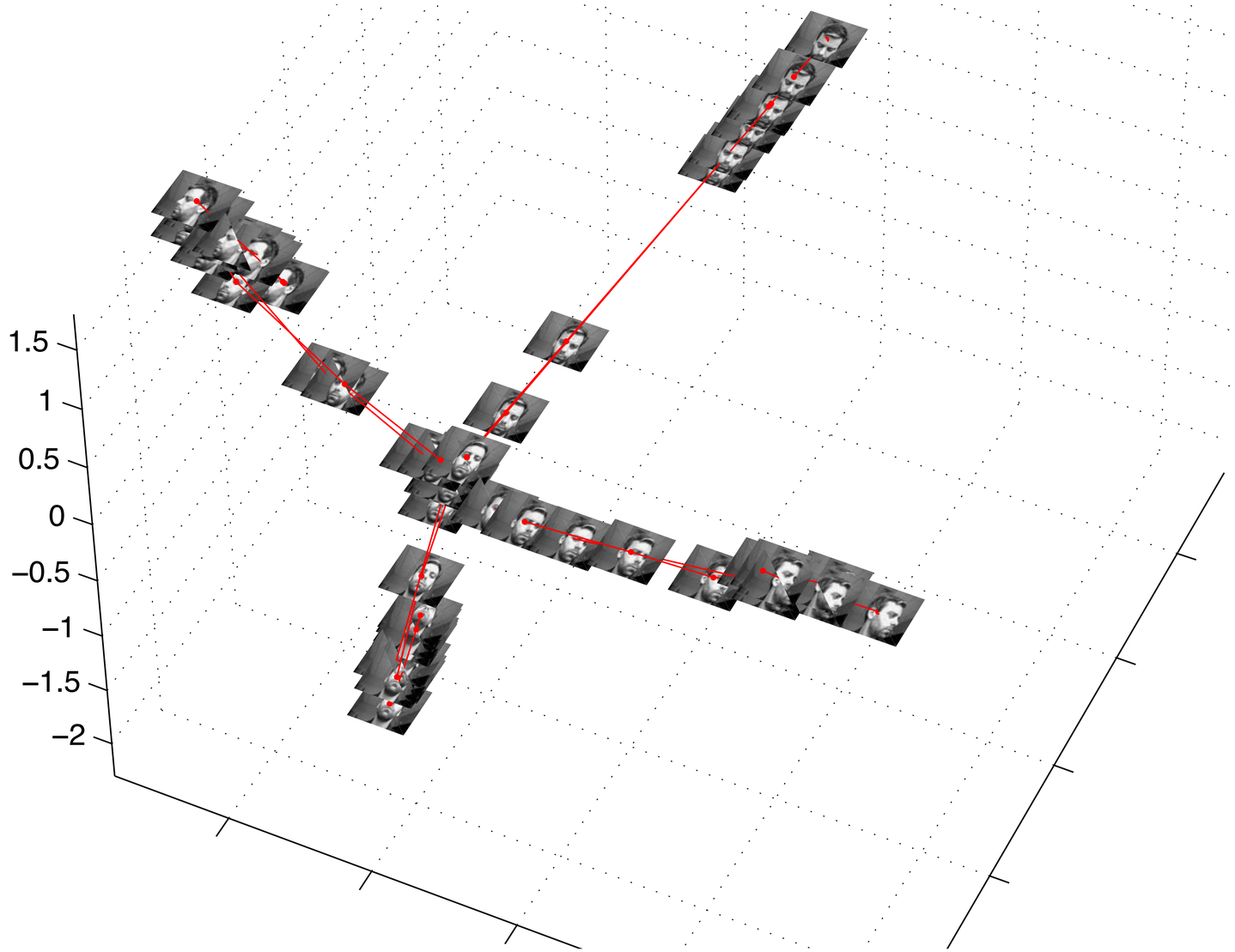
Laplacian Eigenmaps 2D

- Stems coincide with head movements
- Missing dimension
  - Looking up



Laplacian Eigenmaps 3D

- Better!



The usual question

- Which method do I use?
  - The usual answer: “no good answer”
- There are tons of dimensionality reduction approaches, experimentation pays off!

Recap

- Kernel PCA
  - Map to a non-linear space where things are more appropriate for PCA transforms
- MDS
  - Find an embedding that maintains distances
- Manifolds
  - ISOMAP
  - Locally Linear Embedding
  - Laplacian Eigenmaps

Next lecture

- That's all for unsupervised learning for now!
- Moving to classification and supervised learning
- Starting from matched filtering and moving to more advanced approaches to classify data

Reading

- Textbook section 6.7
- Kernel PCA (optional)
  - http://www.springerlink.com/content/w0t1756772h41872/
- Manifolds (optional)
  - http://science.sciencemag.org/content/290/5500/2268
  - http://www.sciencemag.org/cgi/reprint/290/5500/2323.pdf
  - http://www.sciencemag.org/cgi/reprint/290/5500/2319.pdf