CS598PS – Machine Learning for Signal Processing

Dimensionality Reduction – Non-linear Approaches

15 September 2020

Today's lecture

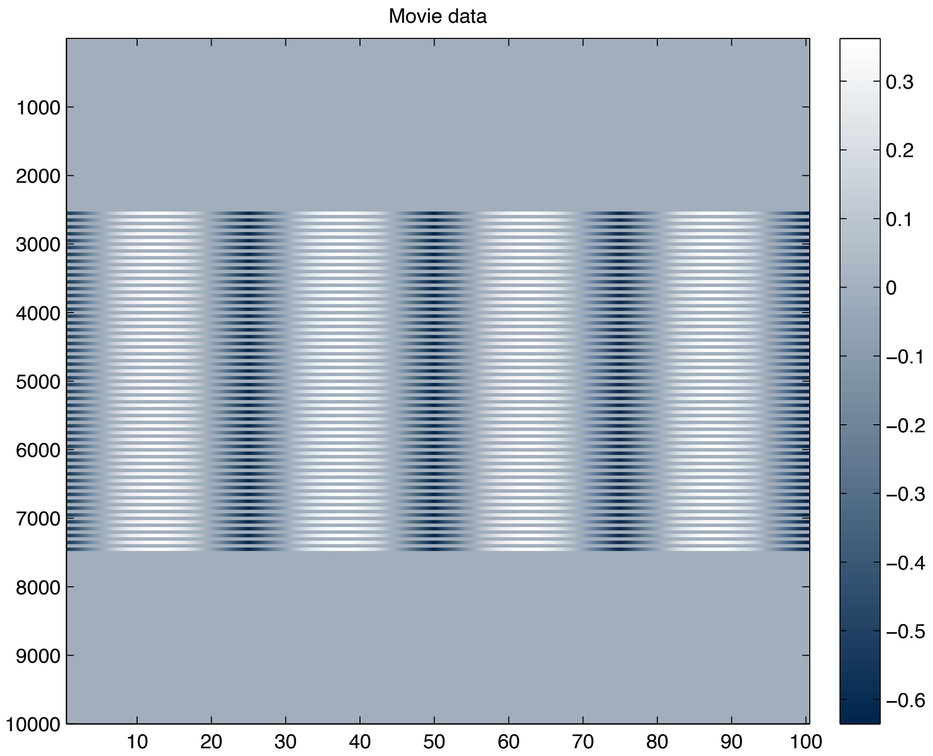
- Non-linear dimensionality reduction
- Kernel PCA
- Manifold Methods

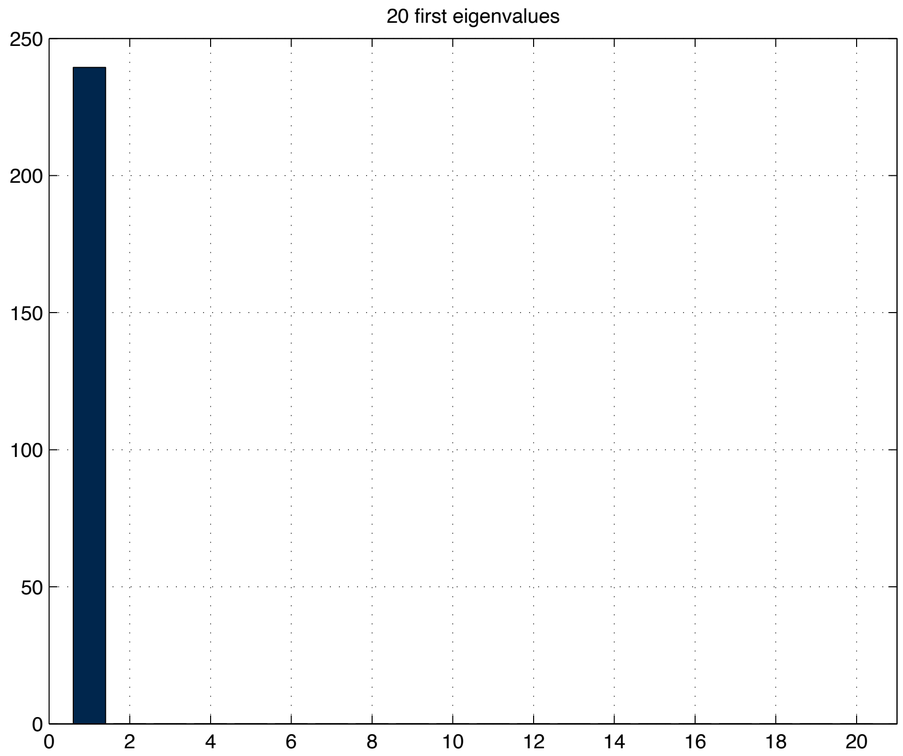
Why dimensionality reduction

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

Very low dimensionality!

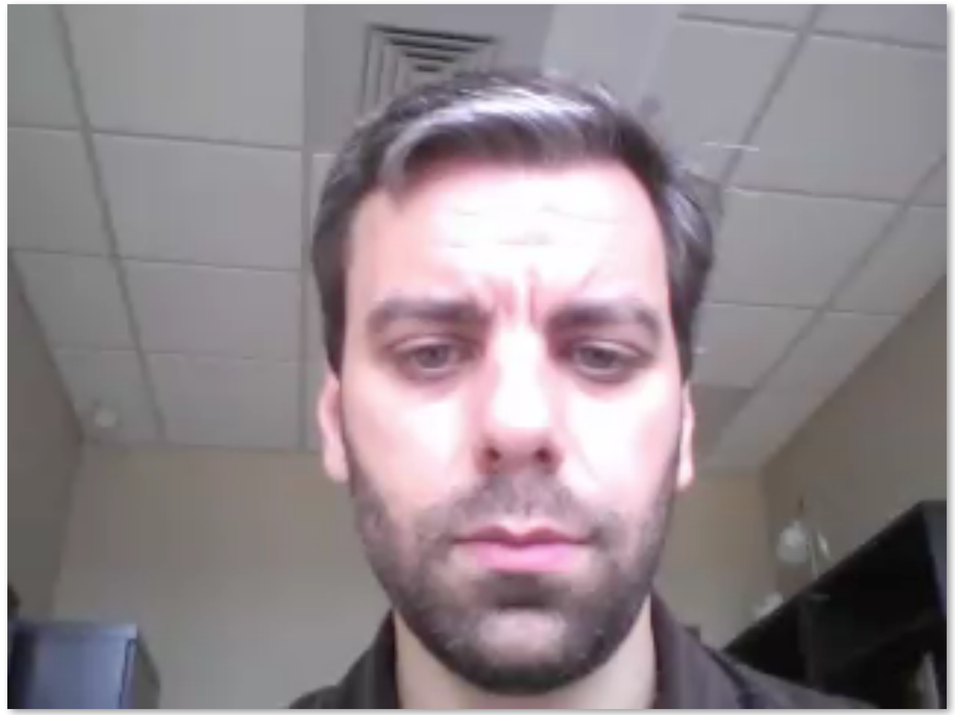
- We effectively need only one pixel to express the video
  - All other active pixels behave the same





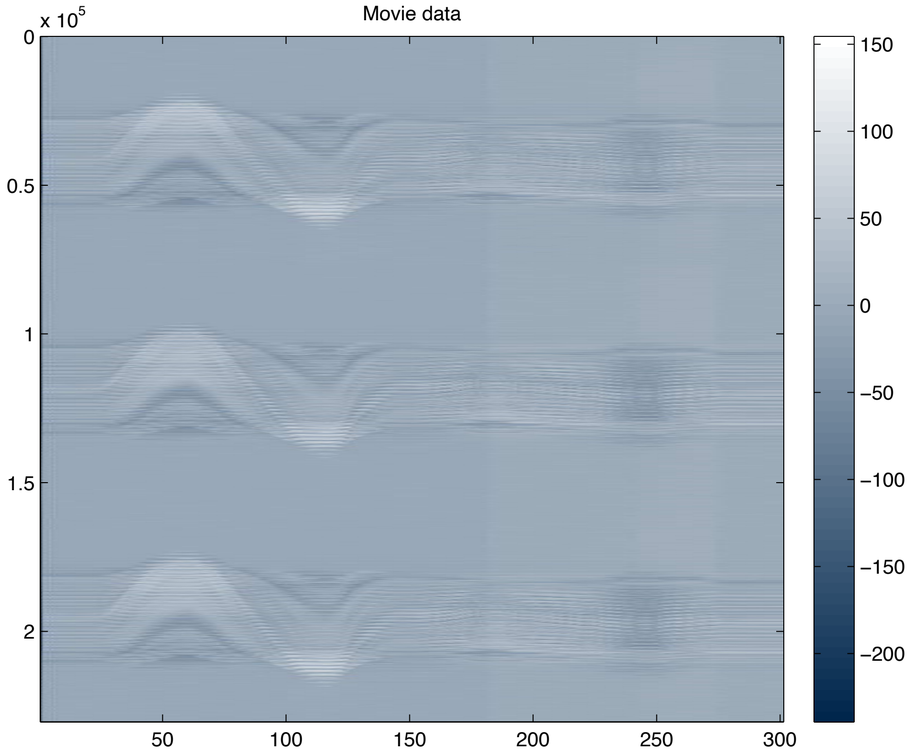
In real life though …

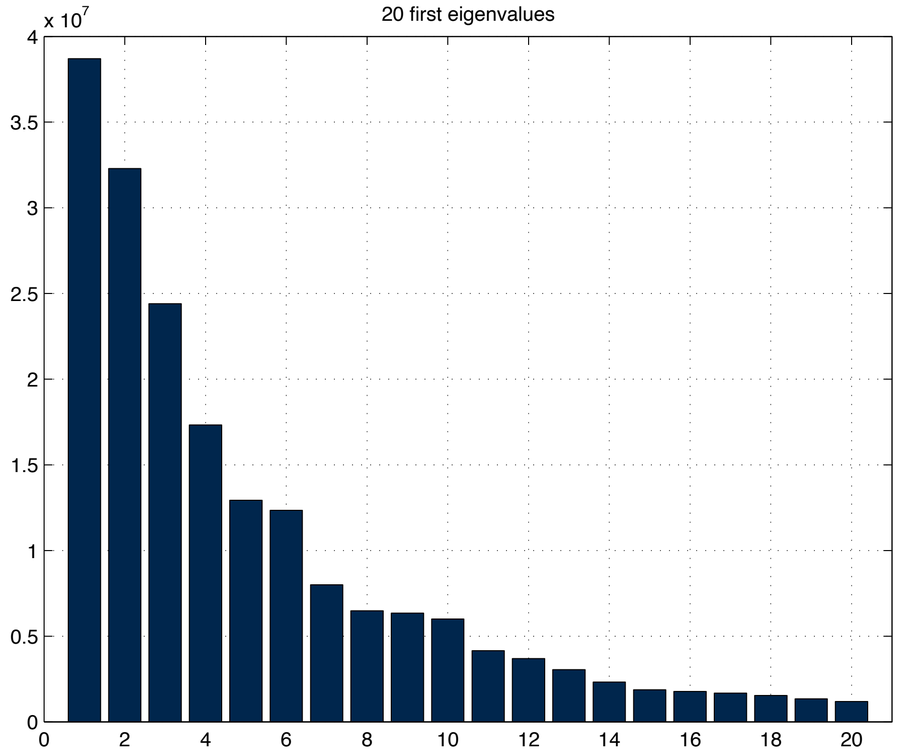
- What is the dimensionality now?
  - Lots of data, $320 \times 240 \times 3 \times 337$



It looks high dimensional

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



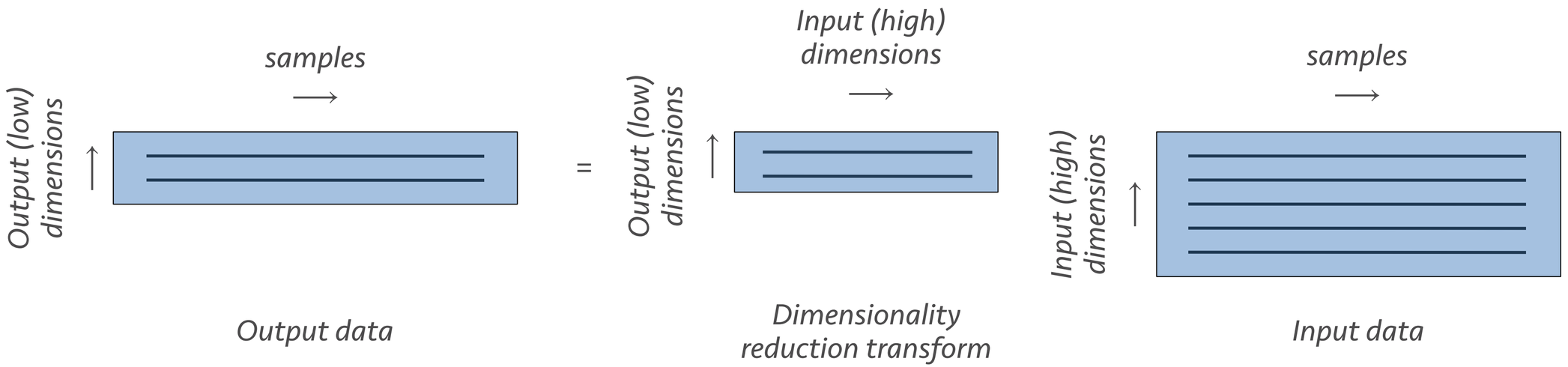


Linear dimensionality reduction

- Linear transform to drop dimensions

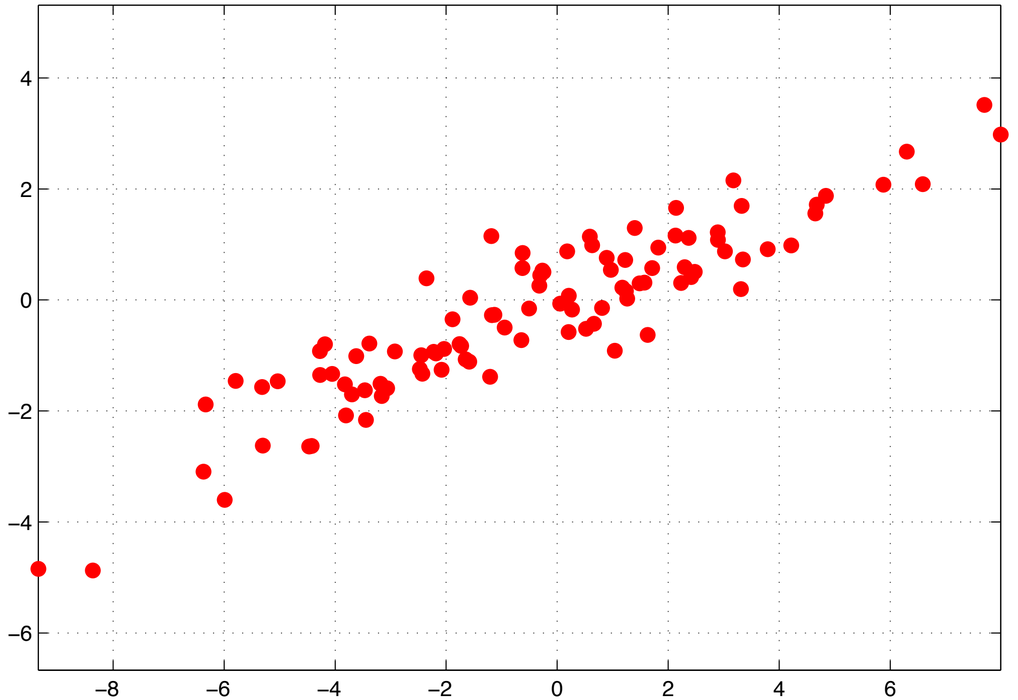
$$\mathbf{Y} = \mathbf{W} \cdot \mathbf{X}$$

  - We saw this with PCA/NMF



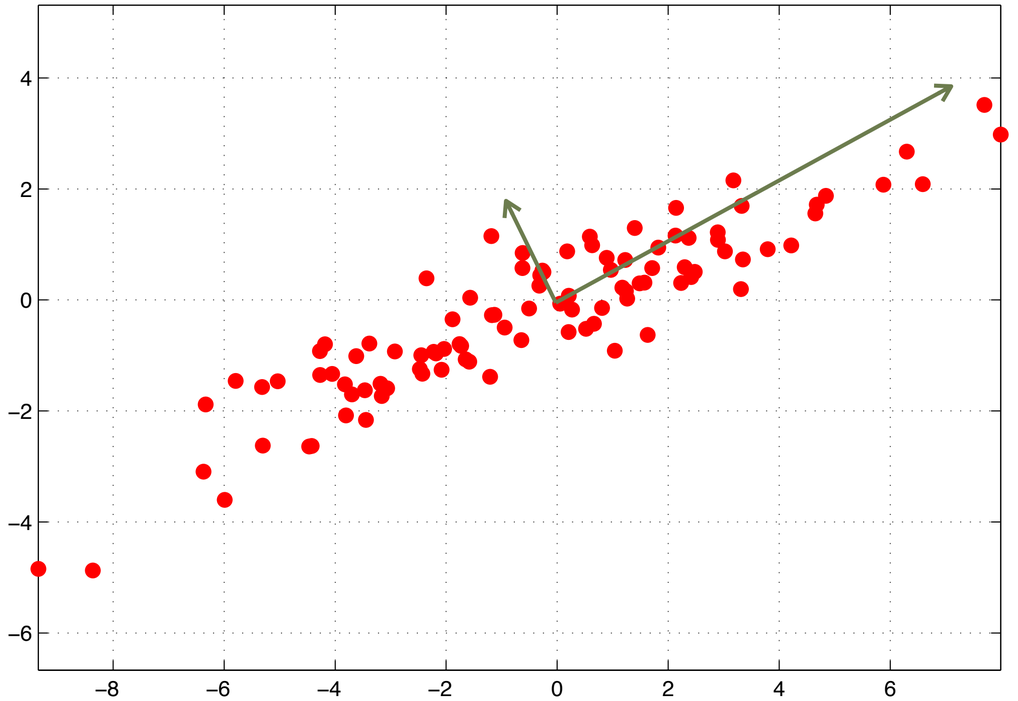
“Easy” data

- Linear transforms work fine with simple data



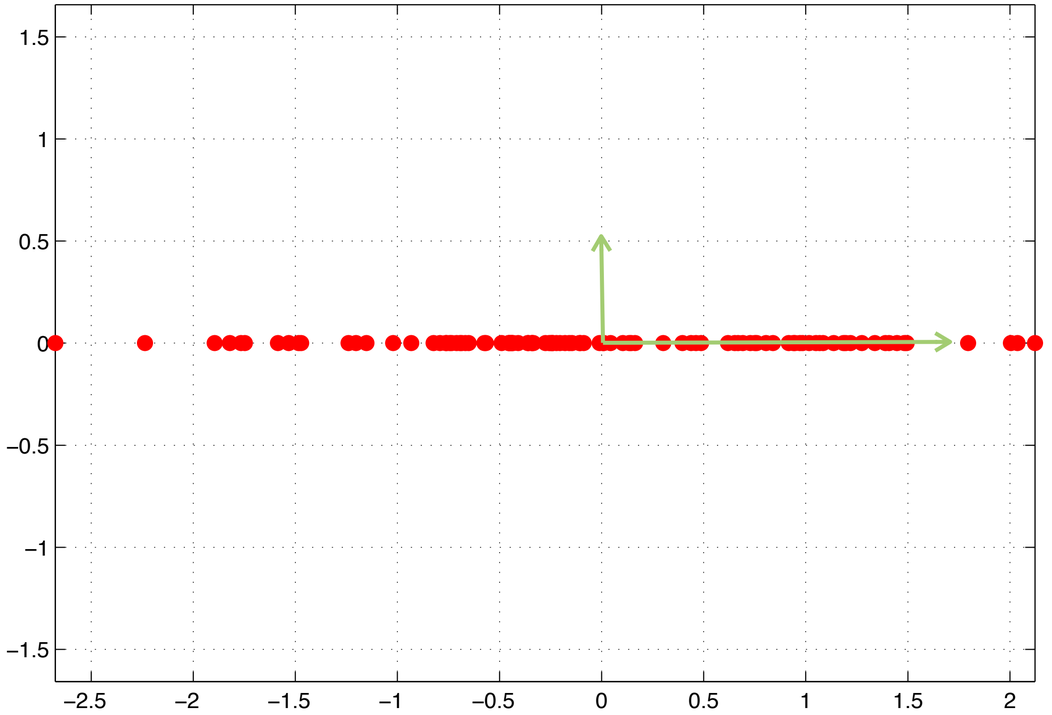
“Easy” data

- Principal components show directions of maximal variance



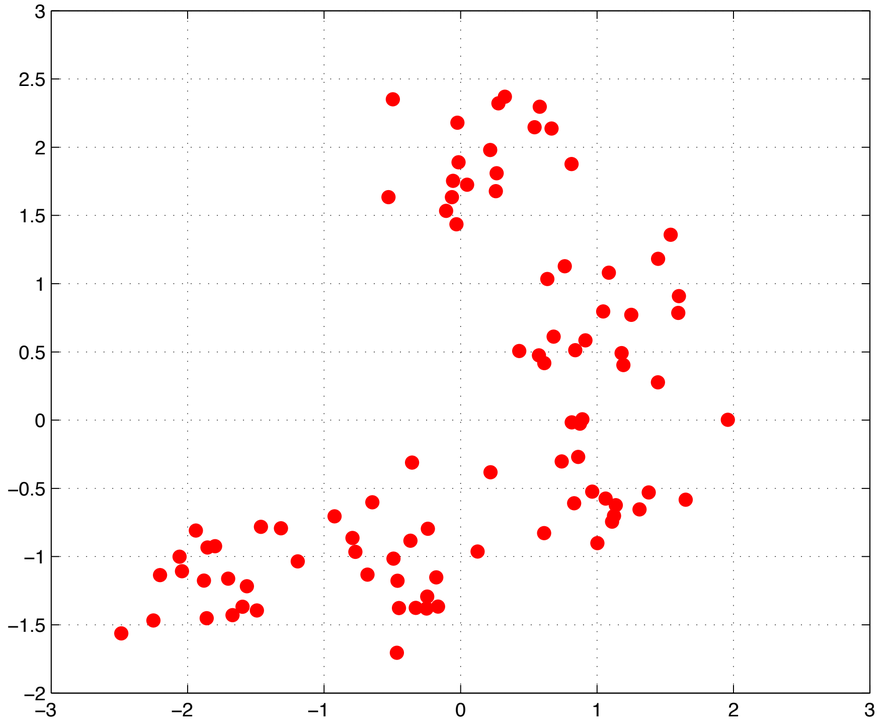
"Easy" data

- Keep only the significant dimensions to reduce dimensionality



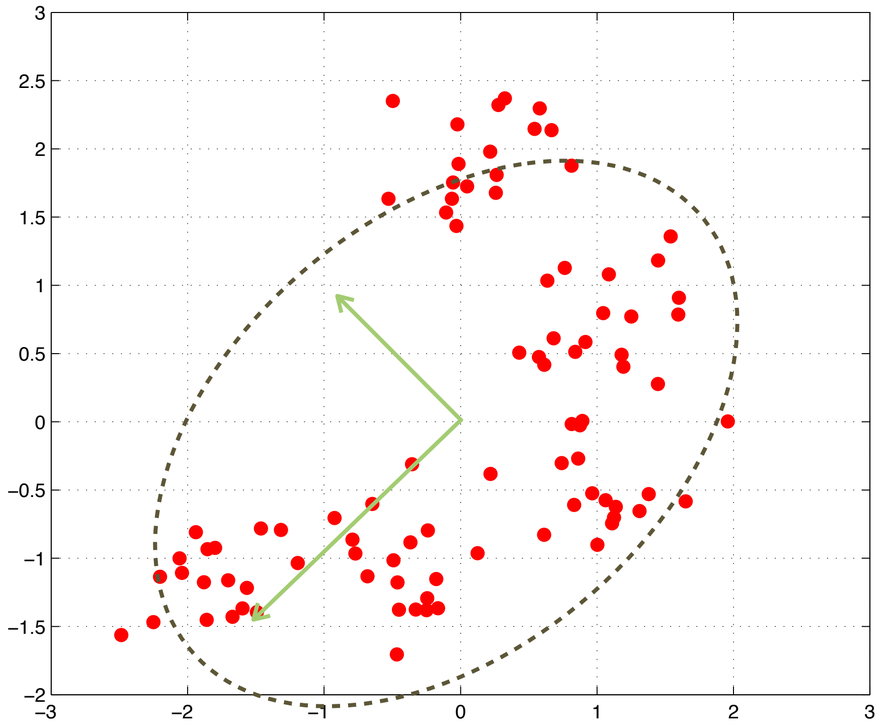
More complicated data

- If data isn't Gaussian(ish) PCA won't be much help



More complicated data

- Not much to interpret here …



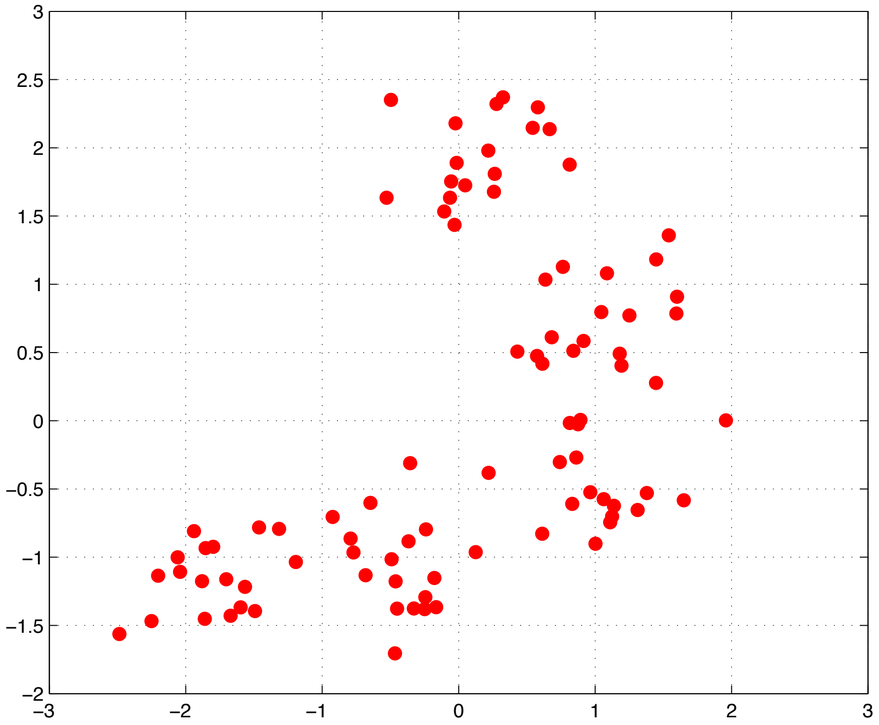
Pinpointing the problem

- 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 other way

- With N, D-dimensional data points

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

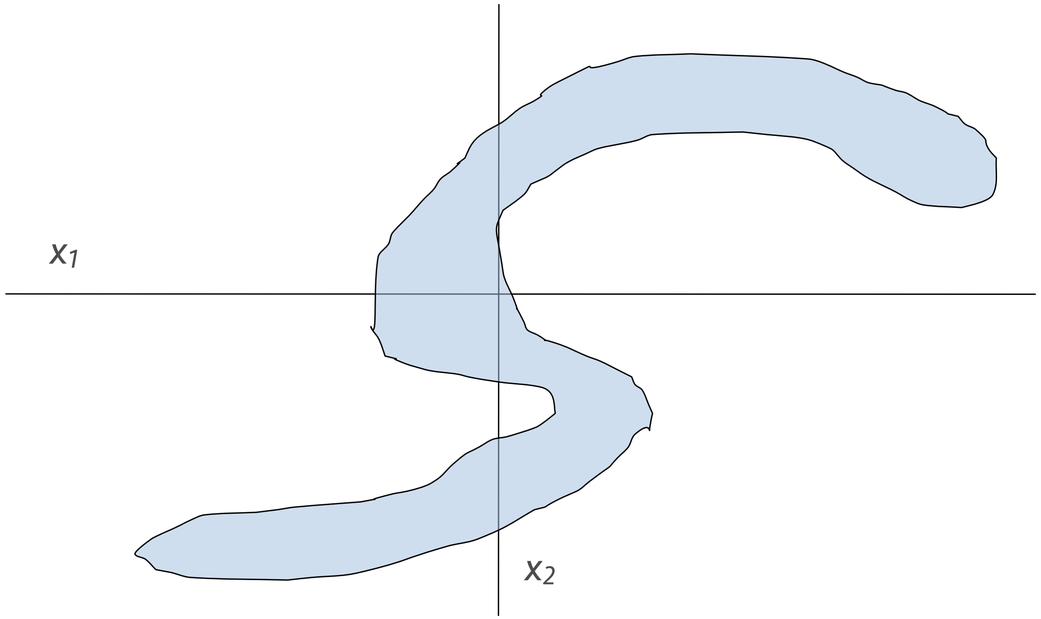
- Assume an unspecified non-linear mapping

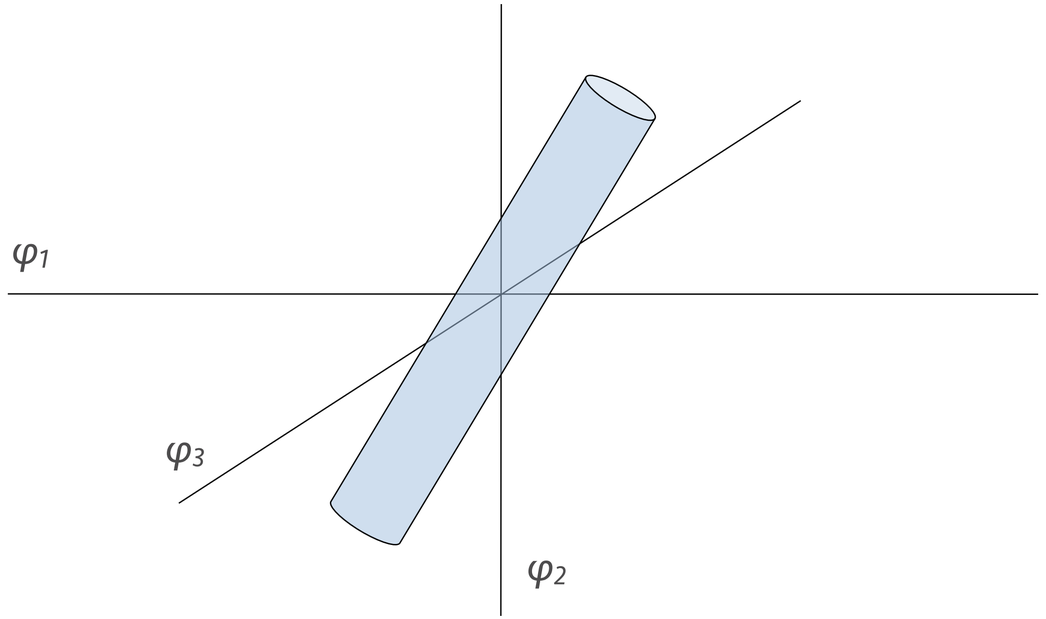
$$\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M, \mathbf{x} \mapsto \mathbf{z} = \phi(\mathbf{x}), M > D$$

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

What we hope for

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





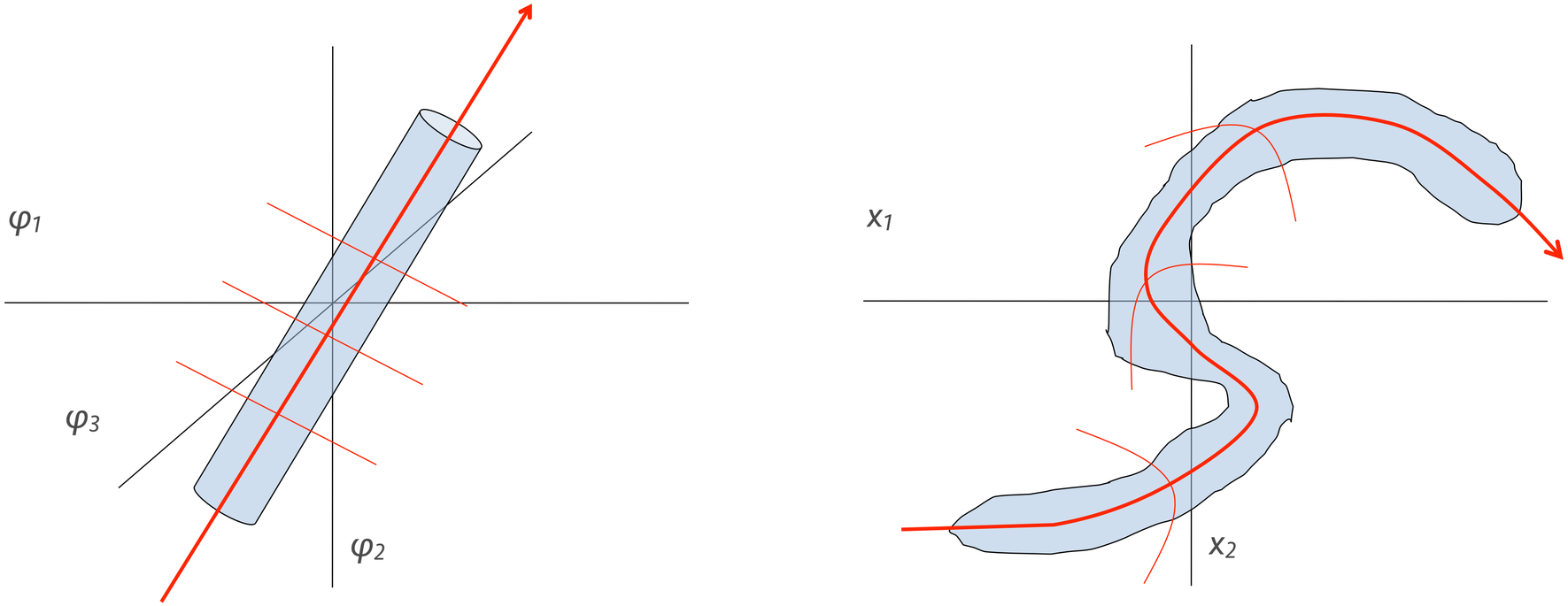
Now we can do PCA!

- With the right mapping, the new data might 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 means slower processing!
    - And increased memory usage

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

$$\mathbf{x} = \begin{bmatrix} x_1, & x_2 \end{bmatrix}^\top \in \mathbb{R}^2, \quad \phi\left(\mathbf{x}\right) = \begin{bmatrix} {x_1}^2, & x_1 x_2, & x_2 x_1, & x_2^2 \end{bmatrix}^\top \in \mathbb{R}^4$$

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!

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

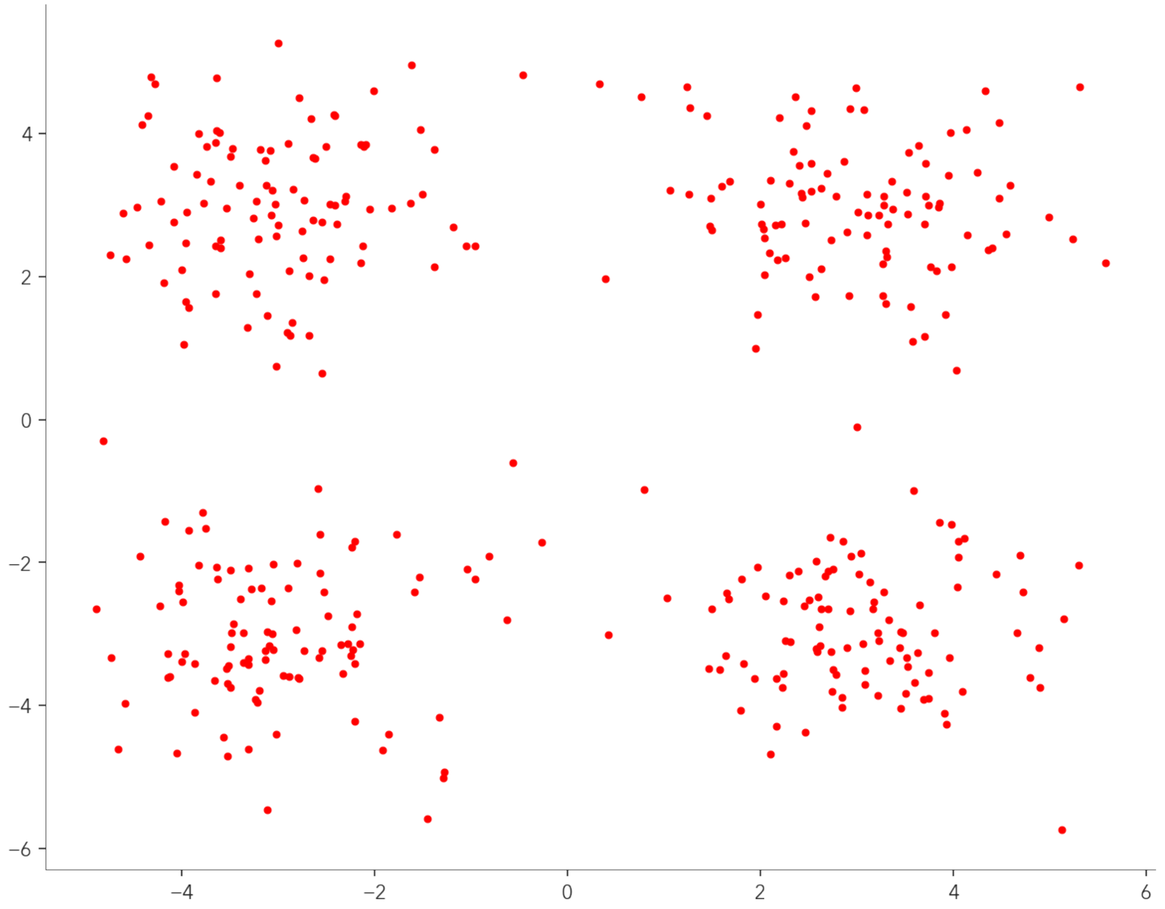
  - The non-linear principal components are a_i
  - The matrix $\mathbf{K}$ is now $N \times N$
- In practice we need to ensure that $\mathbf{K}$ is also positive definite (i.e. $\phi(\mathbf{x})$ is zero mean)
  - Not hard to do, but we skip the details for now

An example

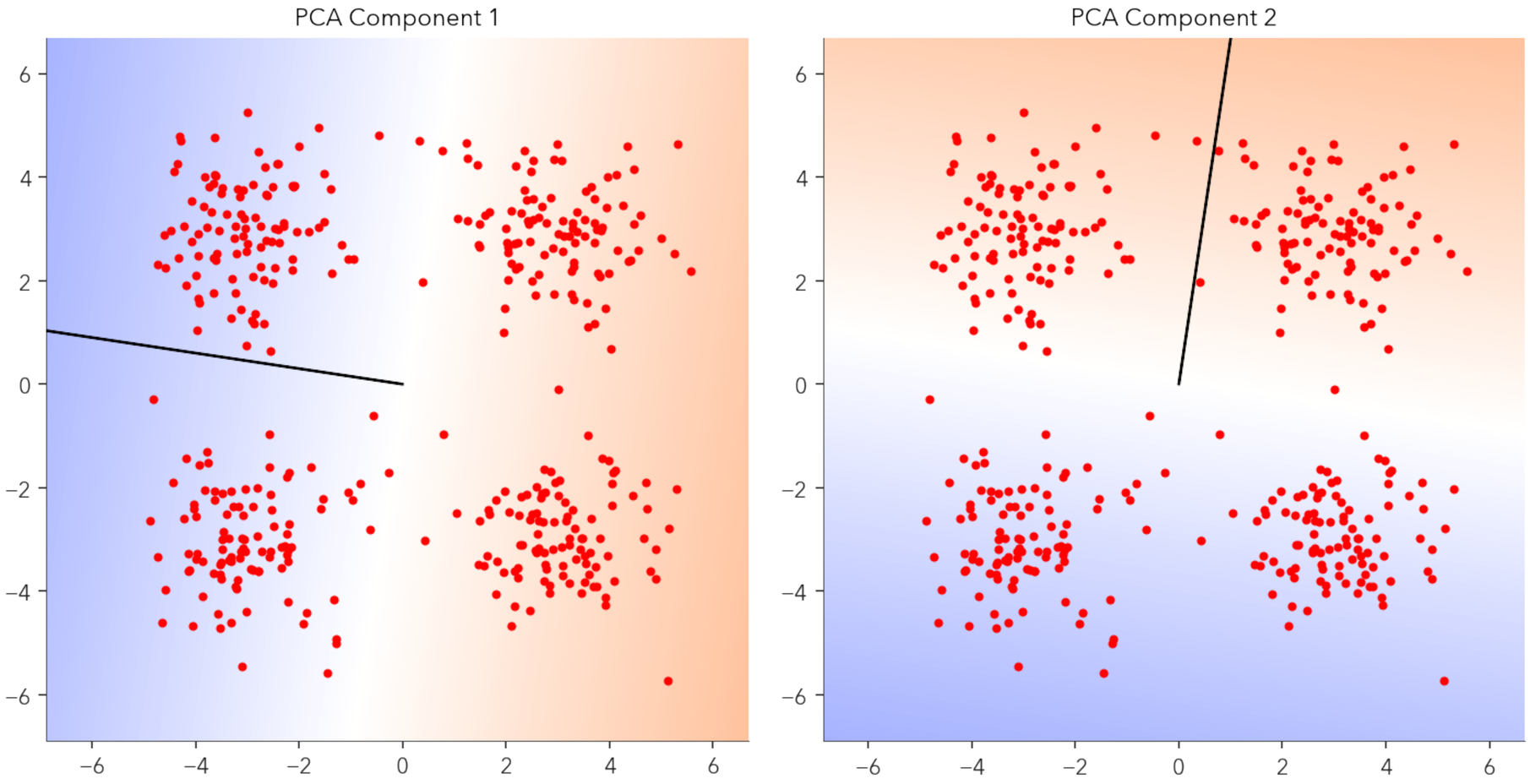
- Data is not good for PCA
  - Isn't Gaussian and has a multimodal distribution
- Use this kernel:

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

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



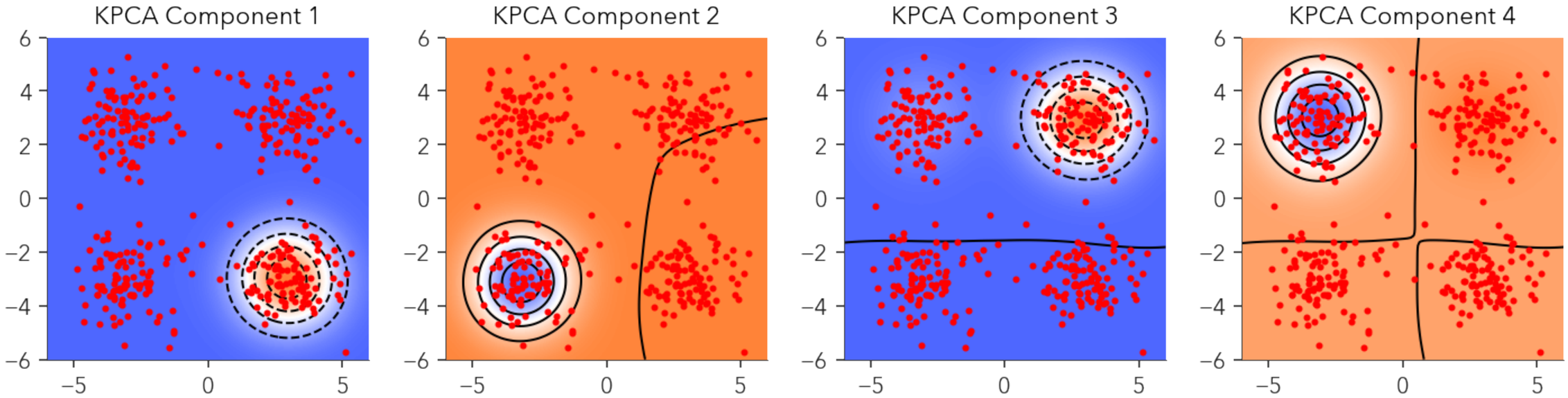
PCA components are not informative



Contour shade shows product of each eigenvector with that (x,y) coordinate: $\mathbf{w}_i^{\top} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$

Kernel PCA components

- Some KPCA advantages
  - Components point to meaningful parts of variation



Contour shade shows product of each eigenvector with kernelized (x,y) coordinate: $\mathbf{w}_i^\top \cdot K\left(\begin{bmatrix} x \\ y \end{bmatrix}, \mathbf{x}\right)$

Varying results depending on the kernel

Laplacian kernel

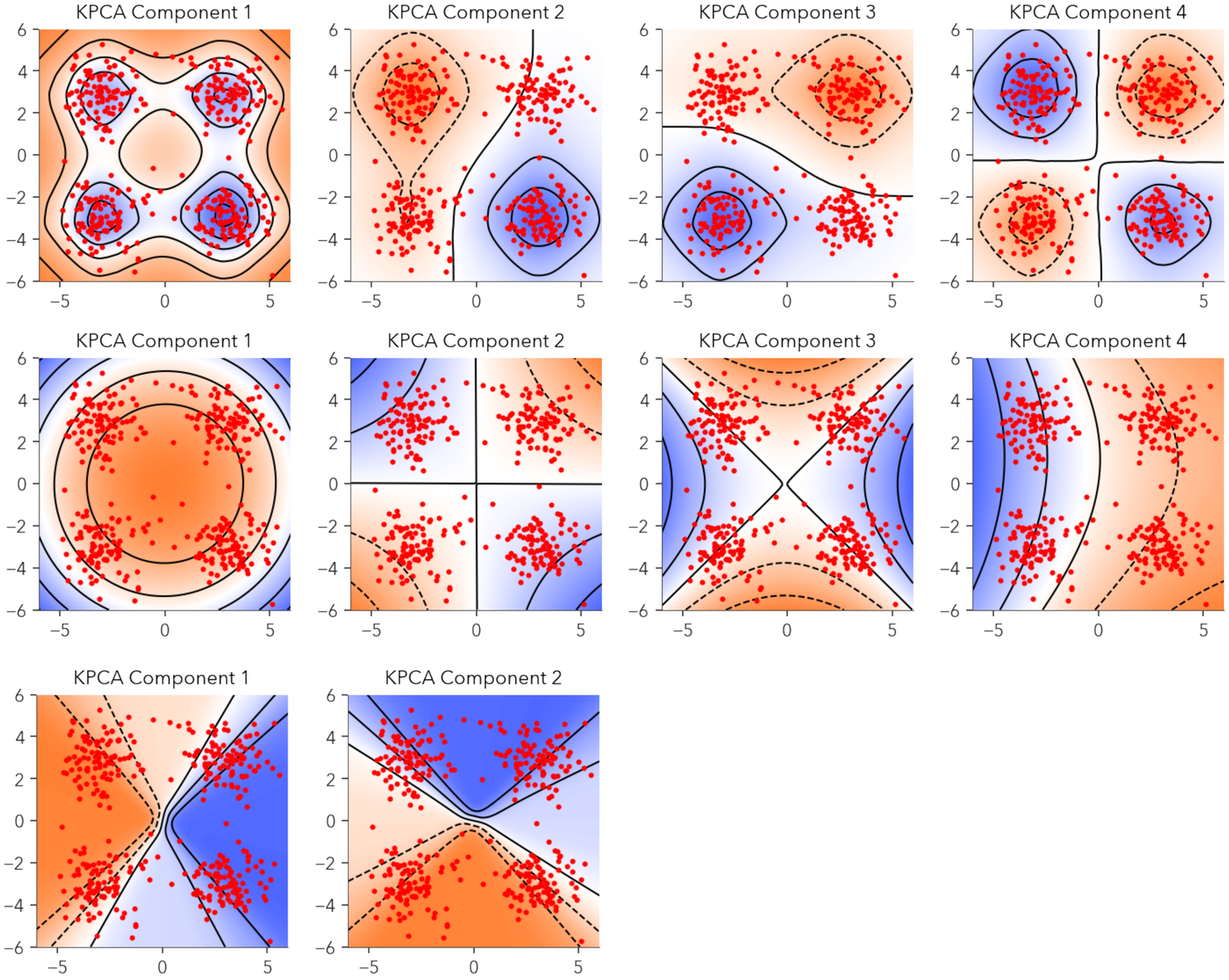
$$K(\mathbf{x},\mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|}{\sigma}}$$

Polynomial kernel

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^\top \cdot \mathbf{y} + 1)^k, k = 2$$

Hyperbolic tangent kernel

$$K(\mathbf{x},\mathbf{y}) = \tanh(x^\top \cdot y + c), c = -1$$



Varying results depending on the kernel

RBF kernel

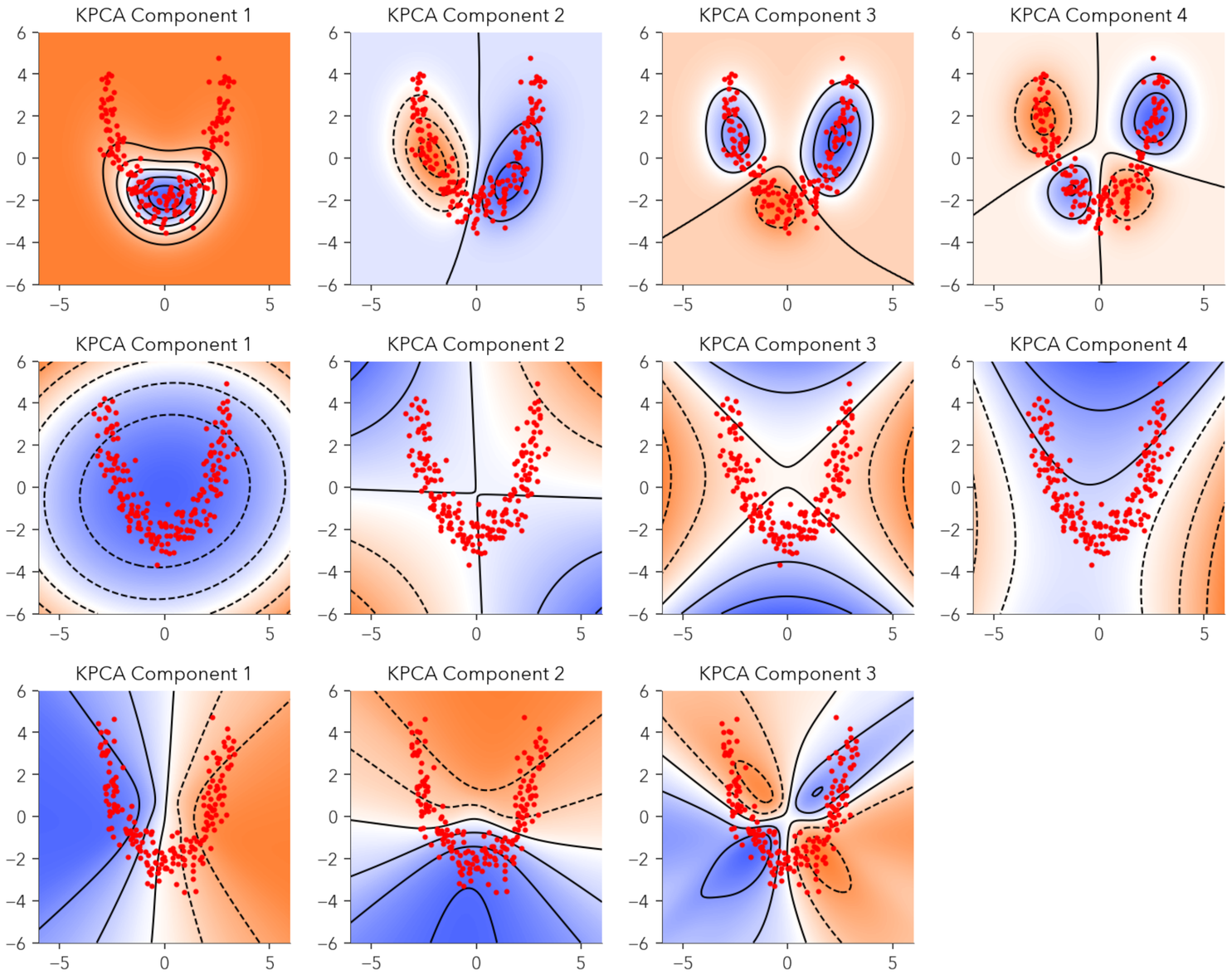
$$K(\mathbf{x},\mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\sigma}}$$

Polynomial kernel

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^\top \cdot \mathbf{y} + 1)^k, k = 2$$

Hyperbolic tangent kernel

$$K(\mathbf{x},\mathbf{y}) = \tanh(x^\top \cdot y + c), c = -1$$



Some KPCA notes

- The eigenanalysis is now bigger
  - $\mathbf{K}$ is $N \times N$
    - Instead of $\mathbf{X} \cdot \mathbf{X}^\top$ we analyze $\phi(\mathbf{X}^\top \cdot \mathbf{X})$
  - So computations will be slower for large data sets
- How do we choose the right kernel?
  - Depends on the data, but you often won't know
  - Try a few

A different take on PCA

- What happens when we only have quantified relationships between our data points?
  - 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

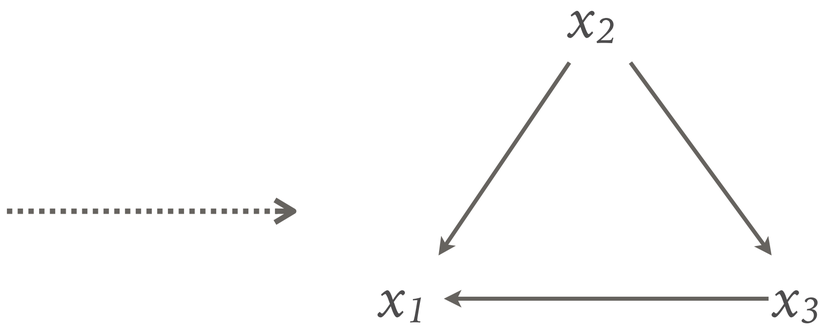
Reconstructing from distances

- 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?

	Tokyo	Stockholm	Singapore	San Francisco	Rome	Rio de Janeiro	Paris	New York City	New Delhi	Moscow	Montreal	Mexico City	Melbourne	London	Honolulu	Hong Kong	Cape Town	Beijing
Tokyo	0.	5092.06	3301.74	5147.83	6140.99	11 533.7	6052.74	6759.99	3638.73	4662.12	6469.09	7033.25	5066.	5957.36	3859.32	1796.24	9154.44	1308.14
Stockholm	5092.06	0.	5991.69	5372.82	1229.06	6637.43	961.227	3938.31	3466.67	765.221	3664.82	5965.1	9685.21	892.676	6871.95	5118.4	6422.11	4177.9
Singapore	3301.74	5991.69	0.	8447.58	6230.8	9778.54	6674.3	9539.	2572.09	5233.2	9203.29	10 327.3	3757.63	6747.1	6727.55	1600.37	6007.23	2773.15
San Francisco	5147.83	5372.82	8447.58	0.	6260.18	6618.08	5578.4	2577.38	7692.26	5885.32	2545.11	1886.71	7855.8	5369.96	2393.62	6908.98	10 248.	5917.89
Rome	6140.99	1229.06	6230.8	6260.18	0.	5704.34	688.964	4291.61	3684.34	1479.03	4100.83	6374.29	9929.85	892.038	8038.73	5775.8	5230.33	5060.38
Rio de Janeiro	11 533.7	6637.43	9778.54	6618.08	5704.34	0.	5682.17	4799.54	8747.88	7164.35	5081.54	4767.82	8220.33	5749.9	8290.28	11 001.2	3774.83	10 763.8
Paris	6052.74	961.227	6674.3	5578.4	688.964	5682.17	0.	3635.36	4102.22	1549.92	3429.48	5722.86	10 431.4	212.534	7448.92	5993.55	5784.08	5117.95
New York City	6759.99	3938.31	9539.	2577.38	4291.61	4799.54	3635.36	0.	7320.53	4680.72	335.712	2087.61	10 362.7	3470.45	4970.19	8068.87	7796.94	6846.19
New Delhi	3638.73	3466.67	2572.09	7692.26	3684.34	8747.88	4102.22	7320.53	0.	2701.46	7010.94	9118.93	6329.5	4178.89	7413.94	2339.32	5768.33	2352.38
Moscow	4662.12	765.221	5233.2	5885.32	1479.03	7164.35	1549.92	4680.72	2701.46	0.	4396.07	6672.02	8954.82	1558.68	7047.73	4443.71	6277.44	3608.11
Montreal	6469.09	3664.82	9203.29	2545.11	4100.83	5081.54	3429.48	335.712	7010.94	4396.07	0.	2316.42	10 397.8	3253.65	4917.32	7739.53	7920.42	6516.64
Mexico City	7033.25	5965.1	10 327.3	1886.71	6374.29	4767.82	5722.86	2087.61	9118.93	6672.02	2316.42	0.	8427.46	5557.29	3786.85	8792.66	8517.05	7751.55
Melbourne	5066.	9685.21	3757.63	7855.8	9929.85	8220.33	10 431.4	10 362.7	6329.5	8954.82	10 397.8	8427.46	0.	10 499.6	5508.65	4593.32	6421.14	5647.46
London	5957.36	892.676	6747.1	5369.96	892.038	5749.9	212.534	3470.45	4178.89	1558.68	3253.65	5557.29	10 499.6	0.	7239.96	5991.01	5987.93	5070.58
Honolulu	3859.32	6871.95	6727.55	2393.62	8038.73	8290.28	7448.92	4970.19	7413.94	7047.73	4917.32	3786.85	5508.65	7239.96	0.	5560.71	11 532.7	5078.94
Hong Kong	1796.24	5118.4	1600.37	6908.98	5775.8	11 001.2	5993.55	8068.87	2339.32	4443.71	7739.53	8792.66	4593.32	5991.01	5560.71	0.	7371.71	1222.89
Cape Town	9154.44	6422.11	6007.23	10 248.	5230.33	3774.83	5784.08	7796.94	5768.33	6277.44	7920.42	8517.05	6421.14	5987.93	11 532.7	7371.71	0.	8042.67
Beijing	1308.14	4177.9	2773.15	5917.89	5060.38	10 763.8	5117.95	6846.19	2352.38	3608.11	6516.64	7751.55	5647.46	5070.58	5078.94	1222.89	8042.67	0.

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:

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

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

- 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 average row/column 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 $\mathbf{D}$:

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

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

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

Multidimensional Scaling (MDS)

- Get distances and "double center":

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

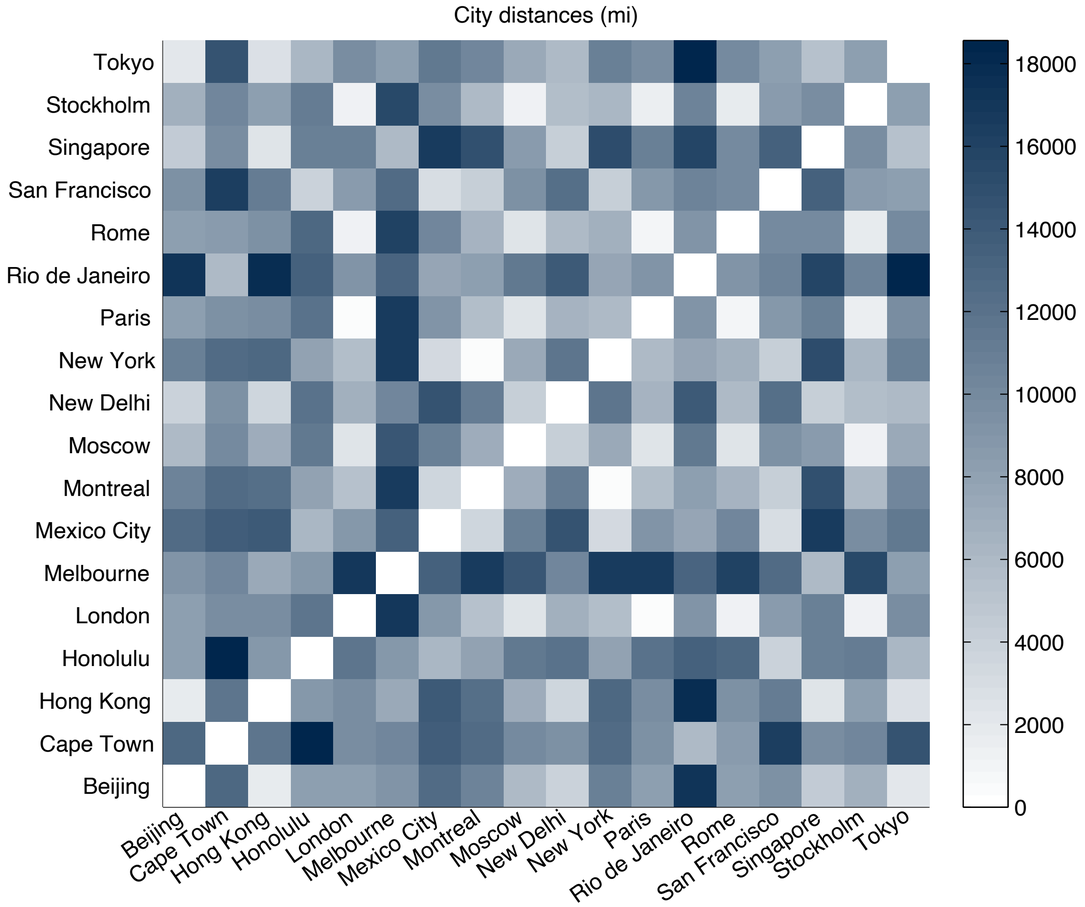
- Estimate $\mathbf{X}$ from $\mathbf{S}$

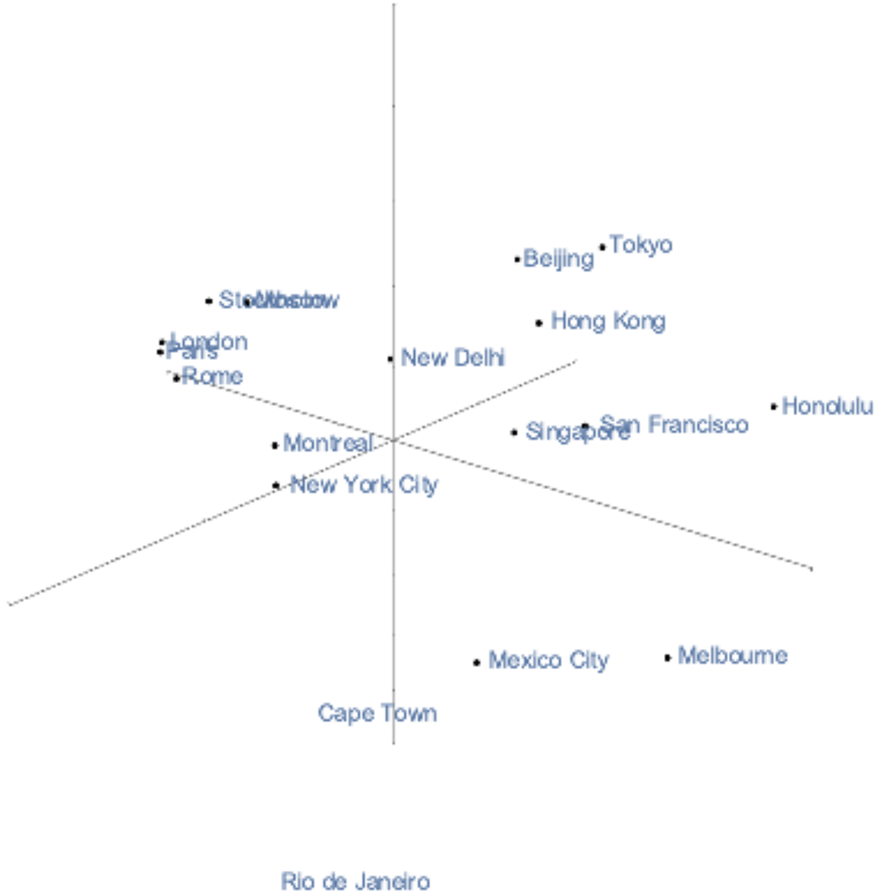
$$\mathbf{S} = \mathbf{X}^\top\cdot\mathbf{X} \Rightarrow \mathbf{S} = \underbrace{\mathbf{U}\cdot\Lambda\cdot\mathbf{U}^\top}_{\textit{Eigendecomposition}} \Rightarrow \hat{\mathbf{X}} = \Lambda^{1/2}\cdot\mathbf{U}^\top$$

- $\mathbf{X}$ 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

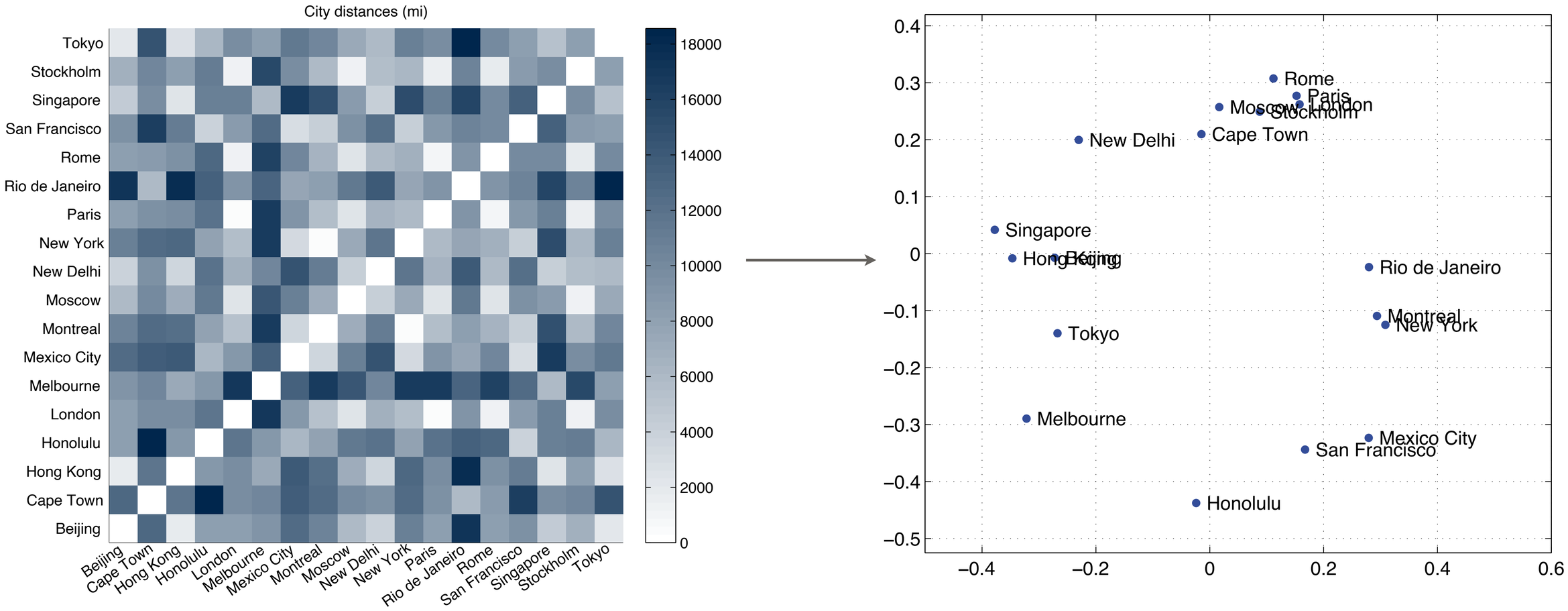
- City distance matrix results in an accurate map!





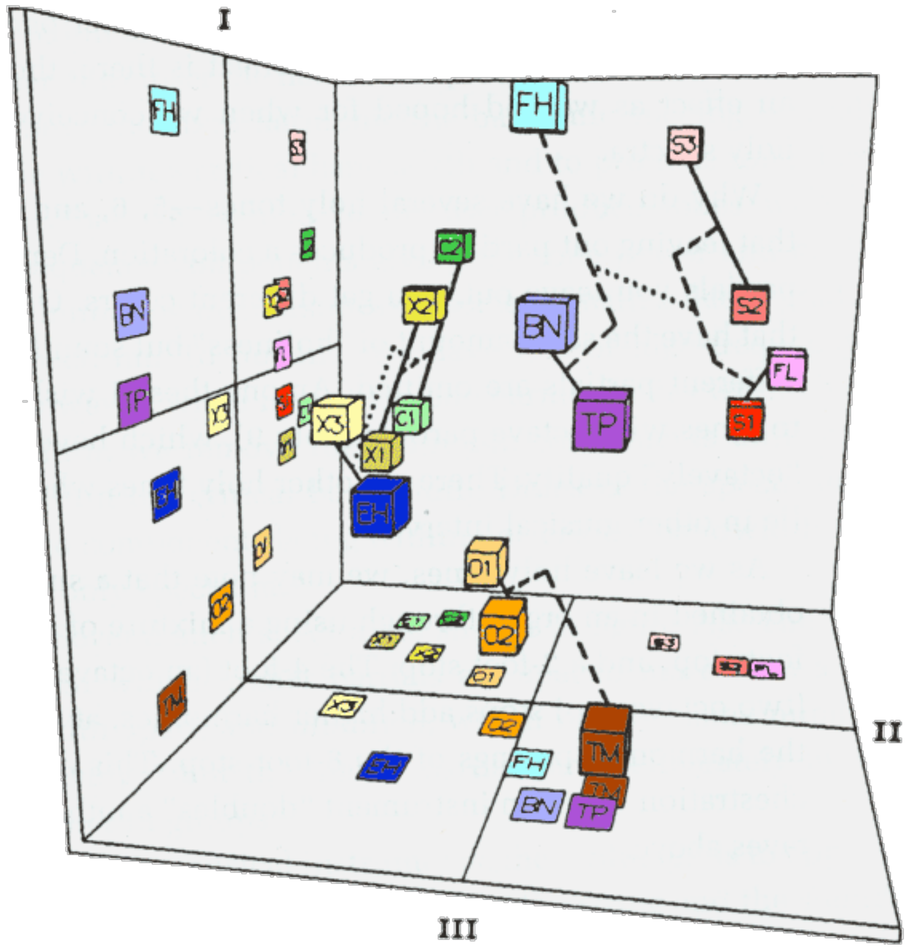
With fewer dimensions

- Also pretty accurate!



Another use

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



MDS variations

- Preserving dot products instead of distances
- Using ranking as instead of 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!

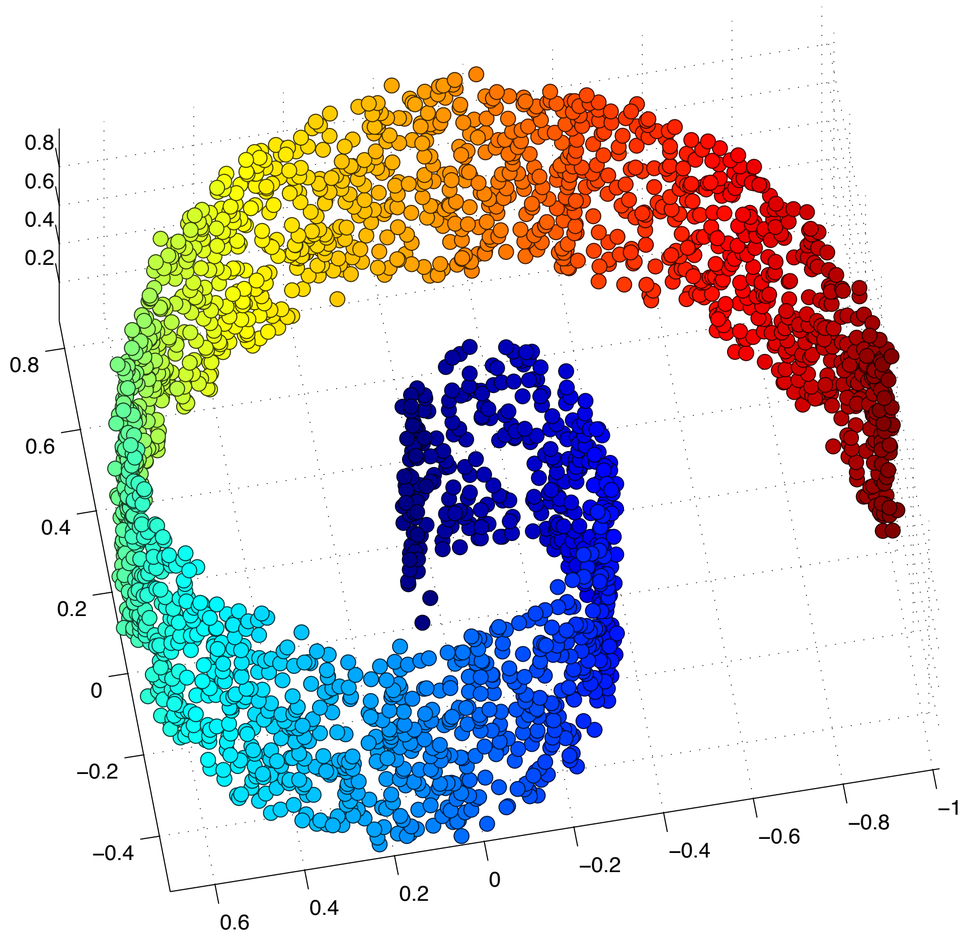
- It is a sort of like PCA
  - Like PCA: But decomposes $\mathbf{X}^\top \cdot \mathbf{X}$ not $\mathbf{X} \cdot \mathbf{X}^\top$
  - Like KPCA: The kernel is just the inner product
  - Is linear

Data with low intrinsic dimension

- Actual dimension can be misleading

Data with low intrinsic dimension

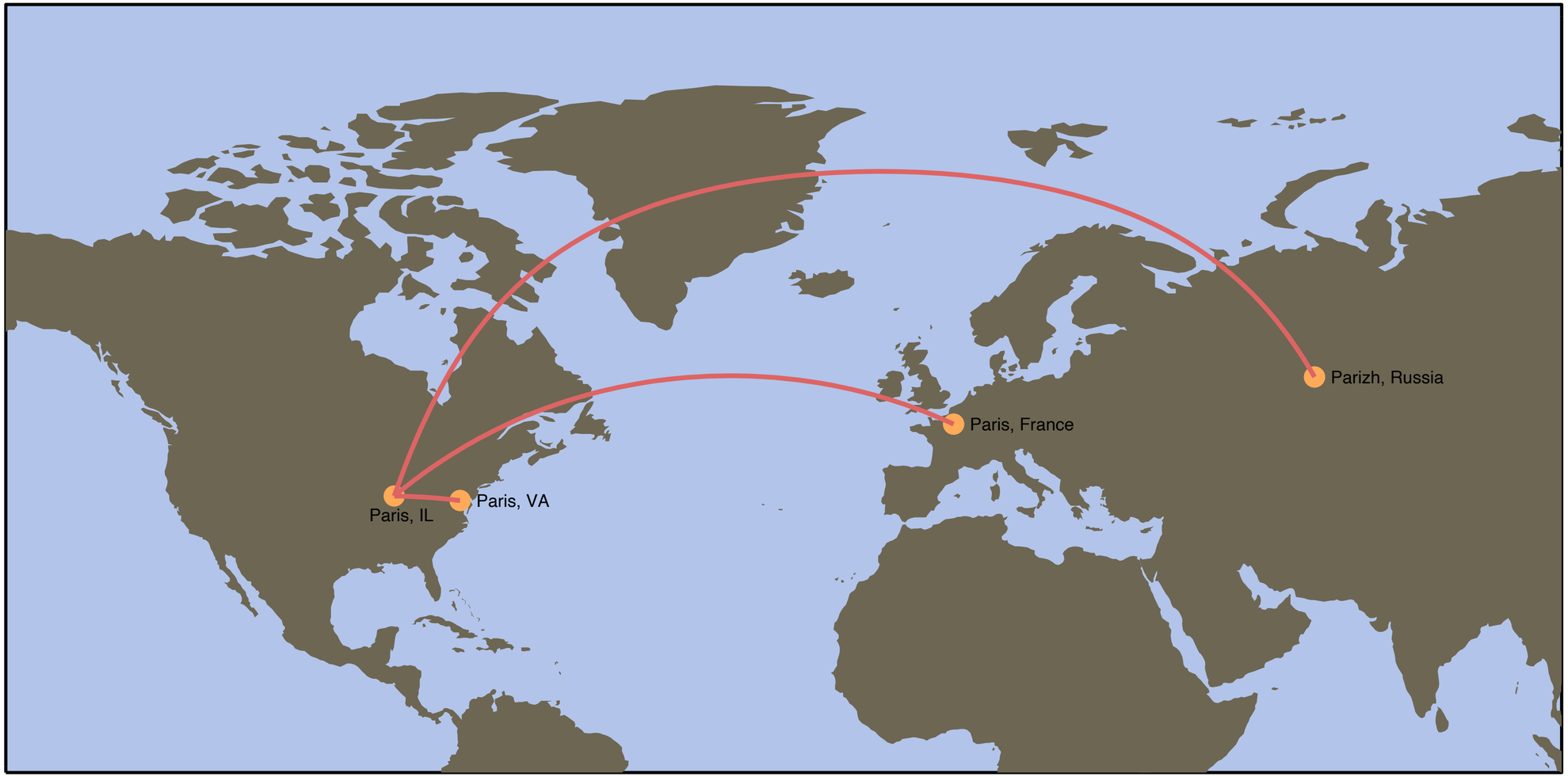
- Actual dimension can be misleading



Local vs. Global

Local vs. Global

Local vs. Global



Preserving topology

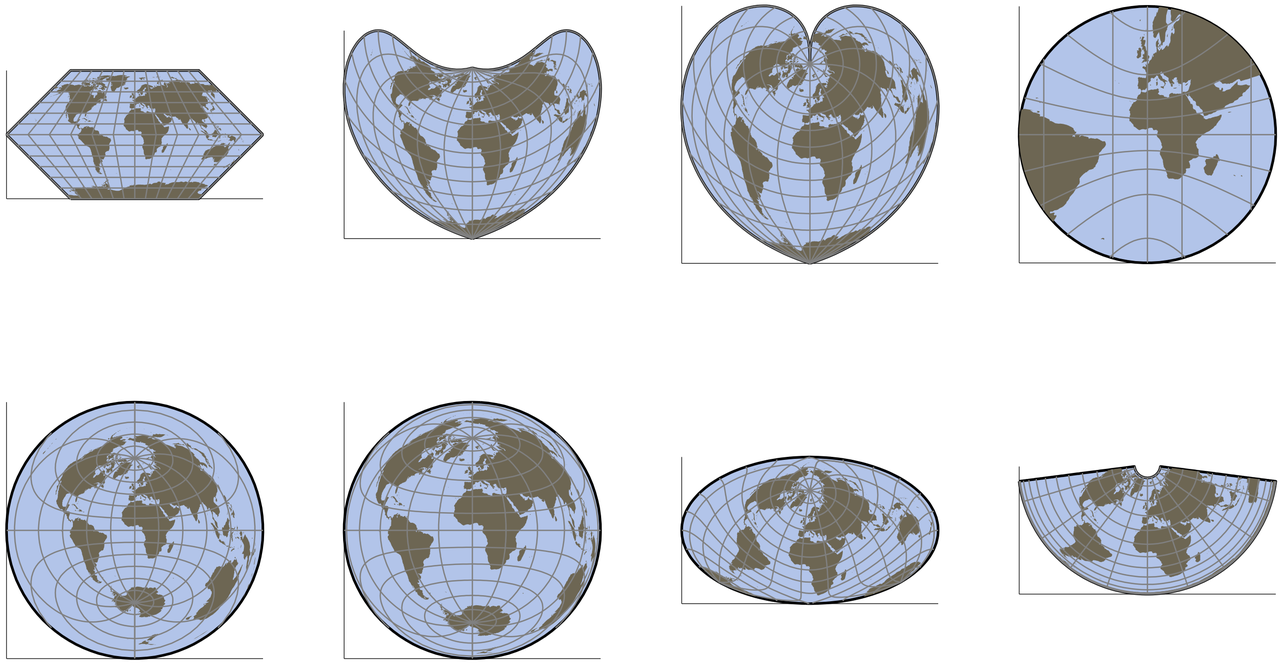
- Looking at the neighboring structure
  - Earth is round, but locally flat

- Reduce dimensionality while keeping the neighborhood structure unchanged

Example: Map projections

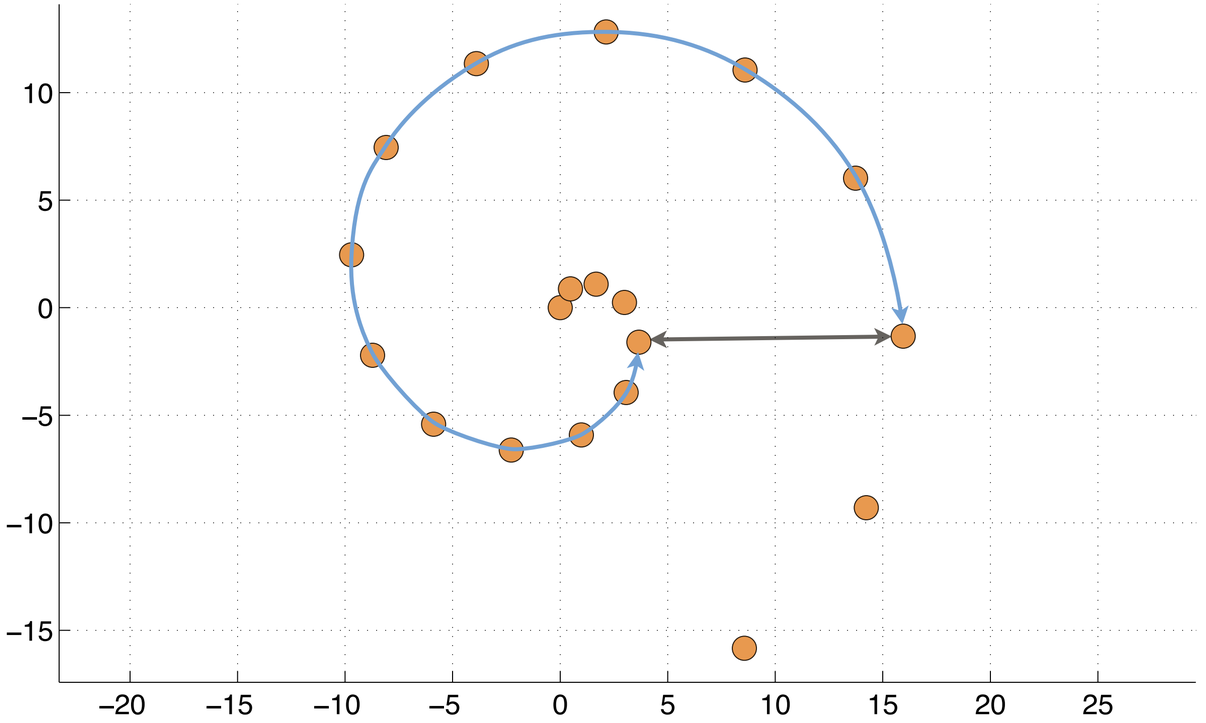
- Many ways to go from 3D to 2D
  - Depends on what you care about
    - How do we do this for data?

Dimensionality reduction



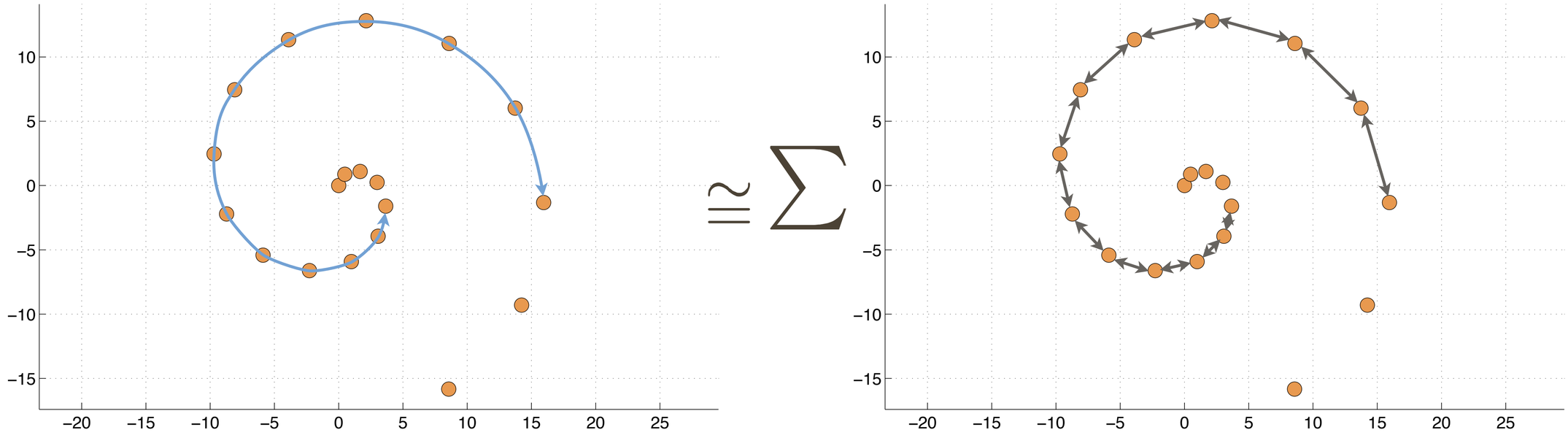
Euclidean vs. Geodesic distance

- On a manifold, distance can mean many things
  - 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

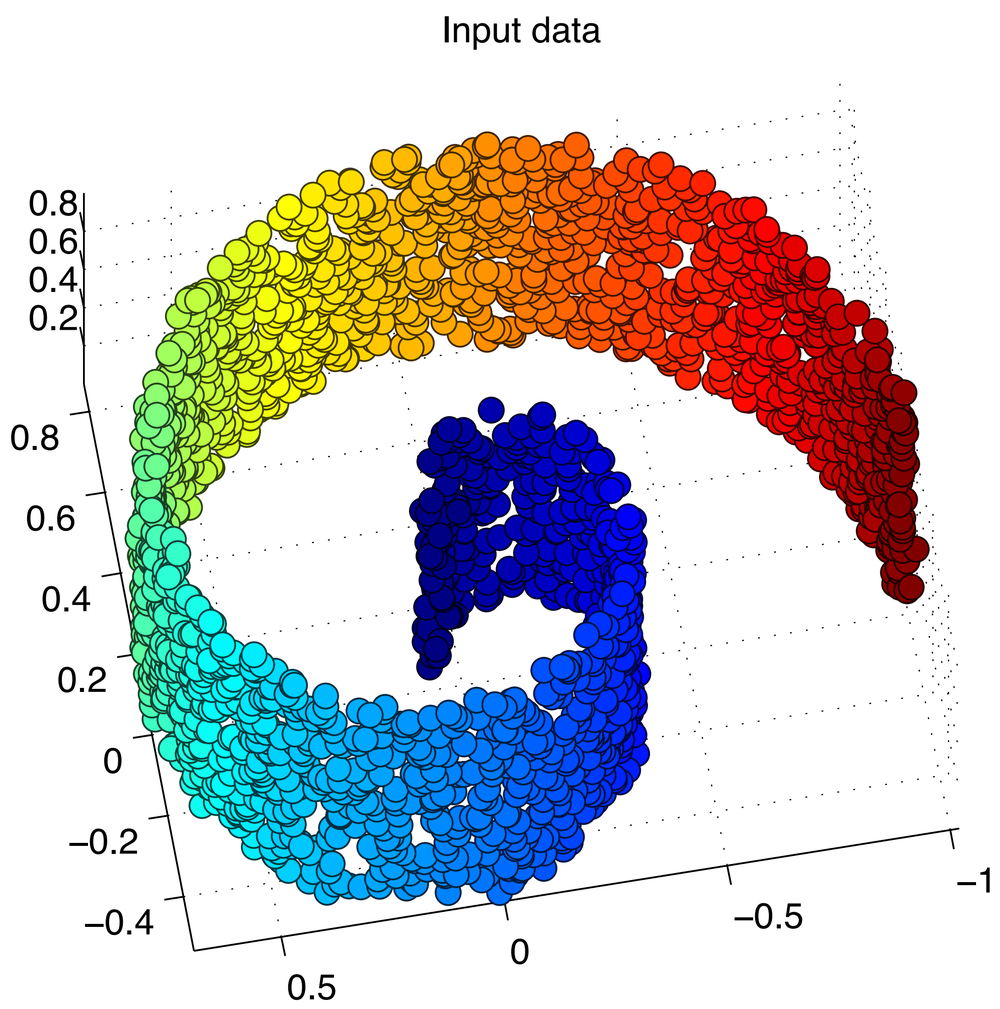


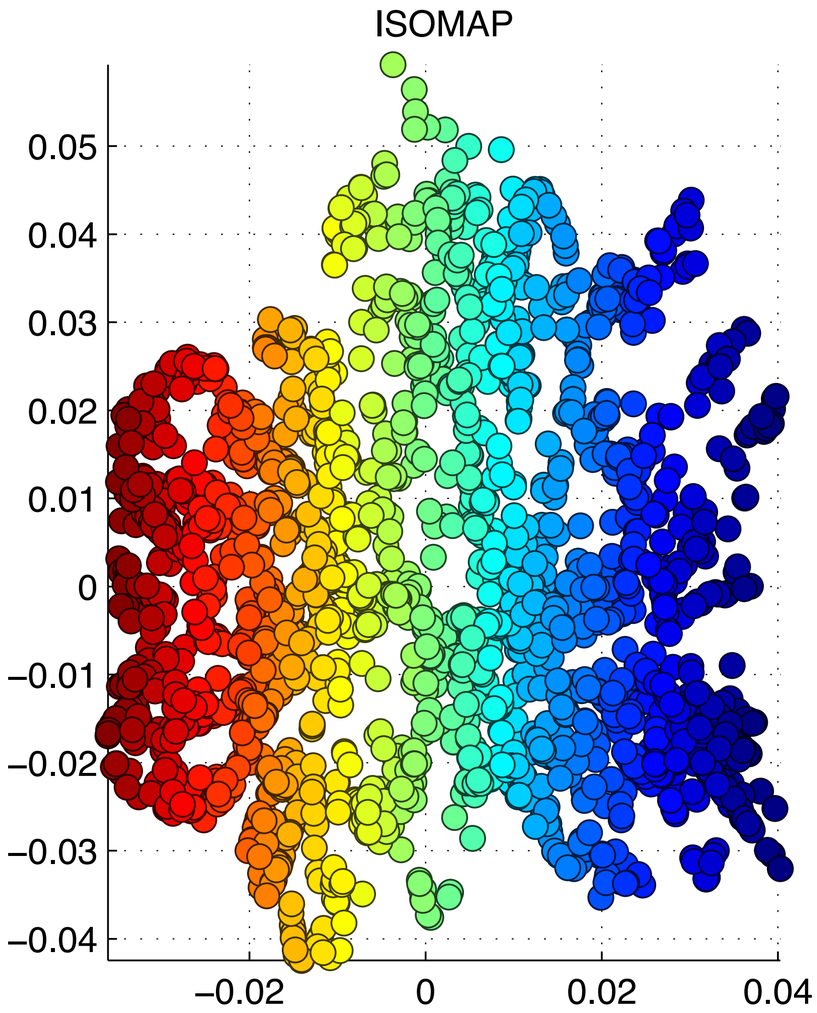
ISOMAP

- Perform MDS using geodesic distances
  - Distance matrix $\mathbf{D}$ now contains geodesic distances

- Results in an embedding that's aware of the low-dimensional structure in our data
  - Projects to Euclidean geometry layout that is proportional to the geodesic distances between data

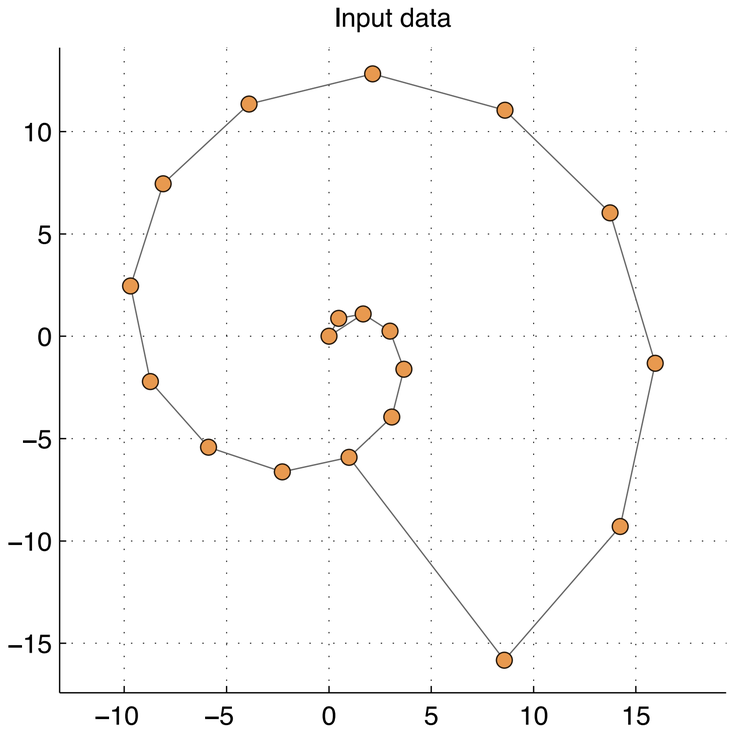
On the Swiss roll

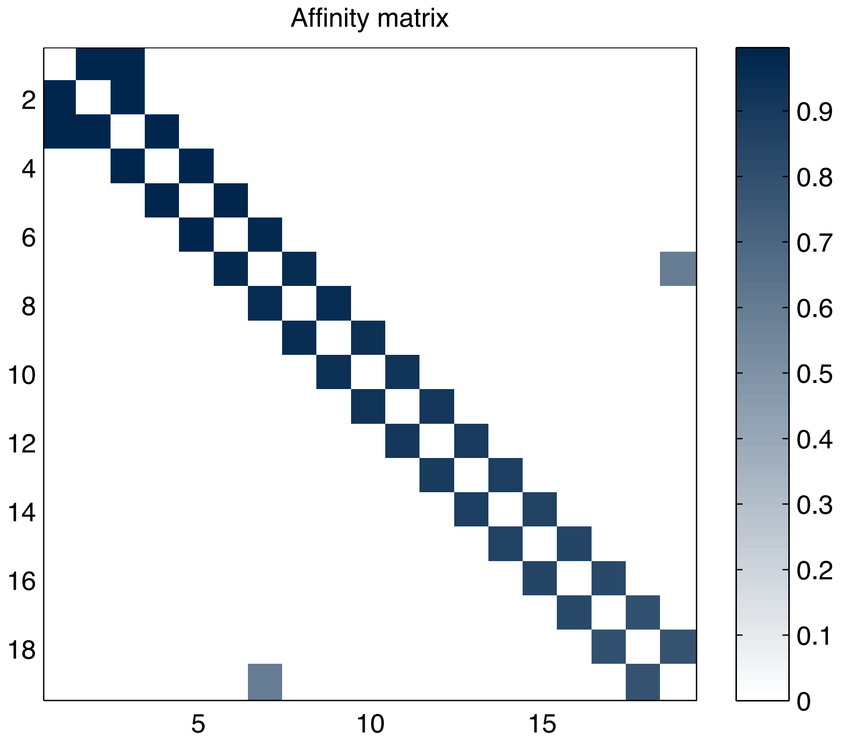




A neighborhood graph approach

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





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

Laplacian Eigenmaps

- 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:

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

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

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

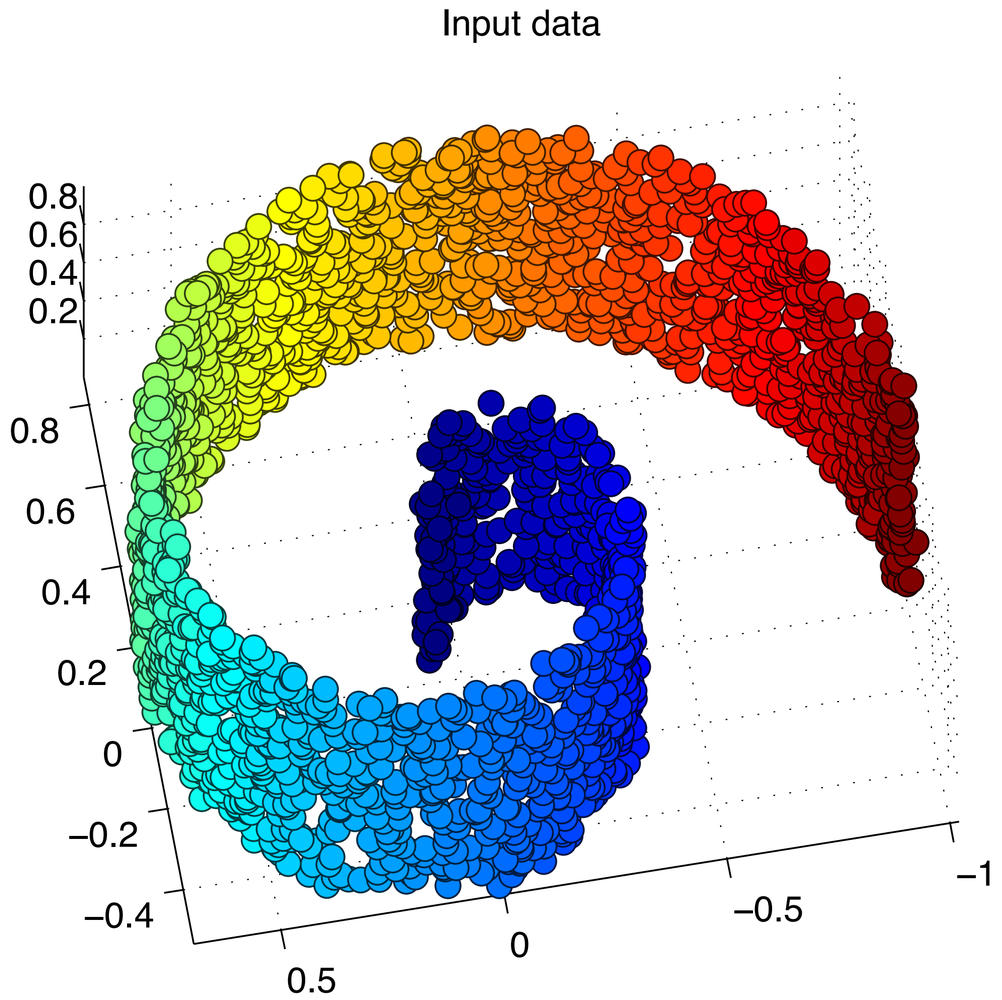
And predictably …

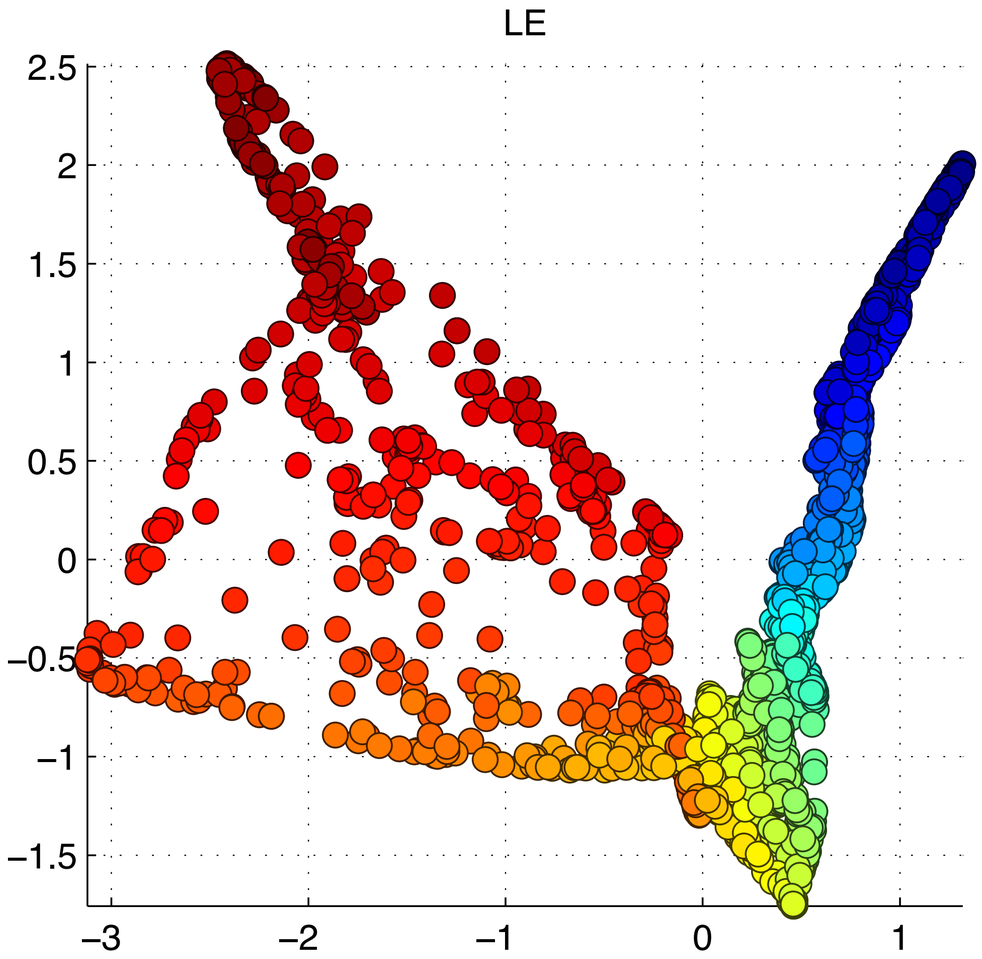
- $\mathbf{Z}$ are the eigenvectors of the normalized Laplacian
  - From which we get $\mathbf{Y}$

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

- The eigenvalues will help us select dimensions of $\mathbf{Y}$
  - 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





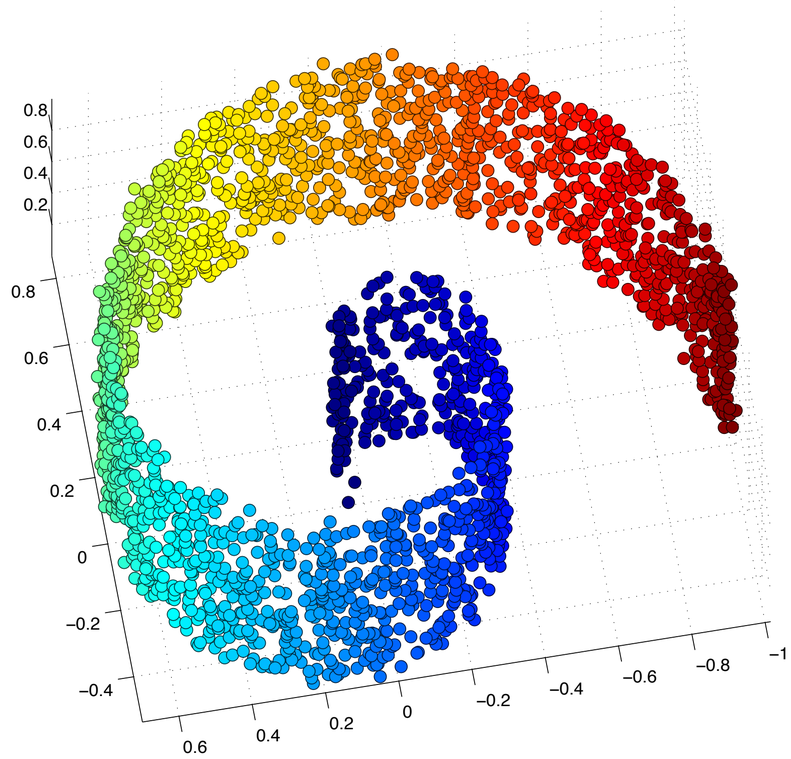
Locally Linear Embedding

- 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 $\mathbf{W}$

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

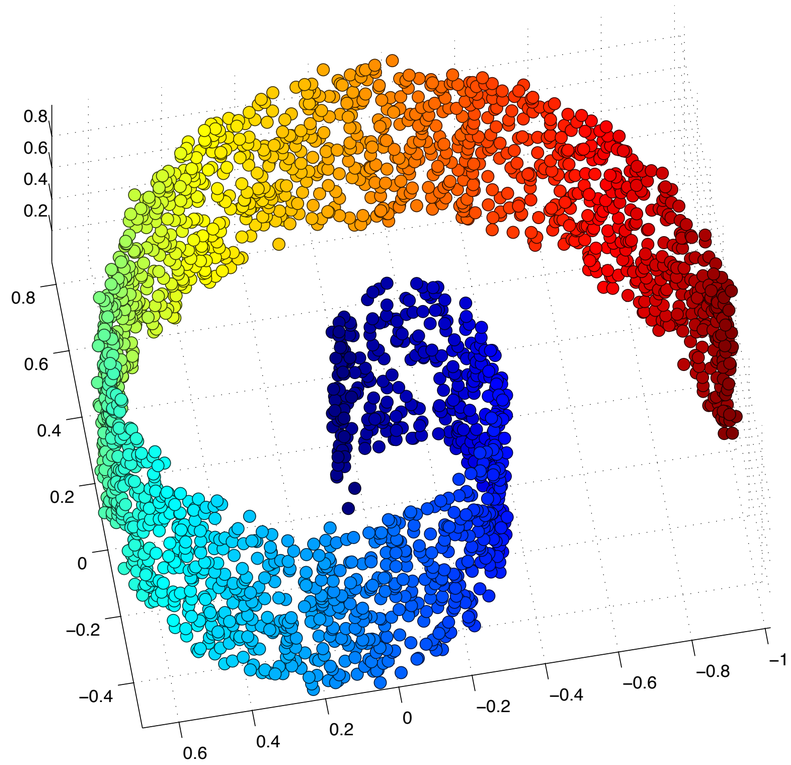


Locally Linear Embedding

- 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

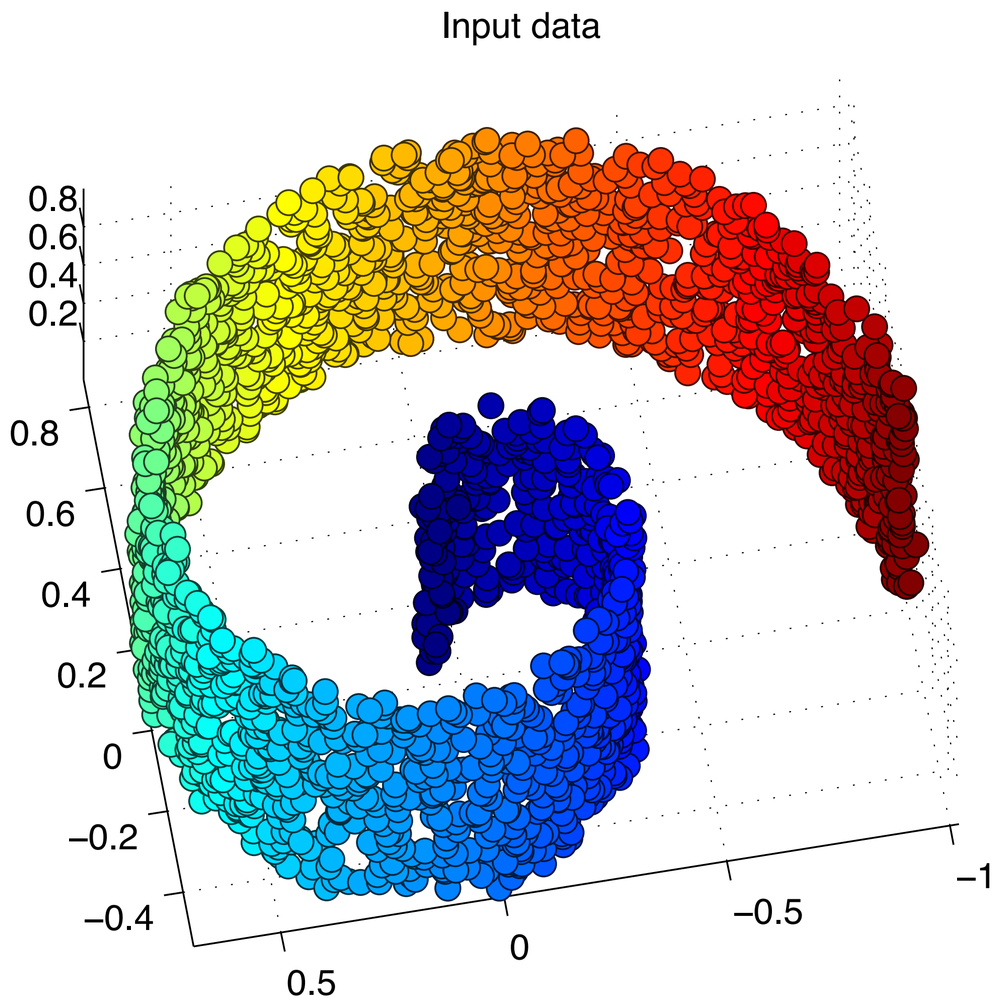


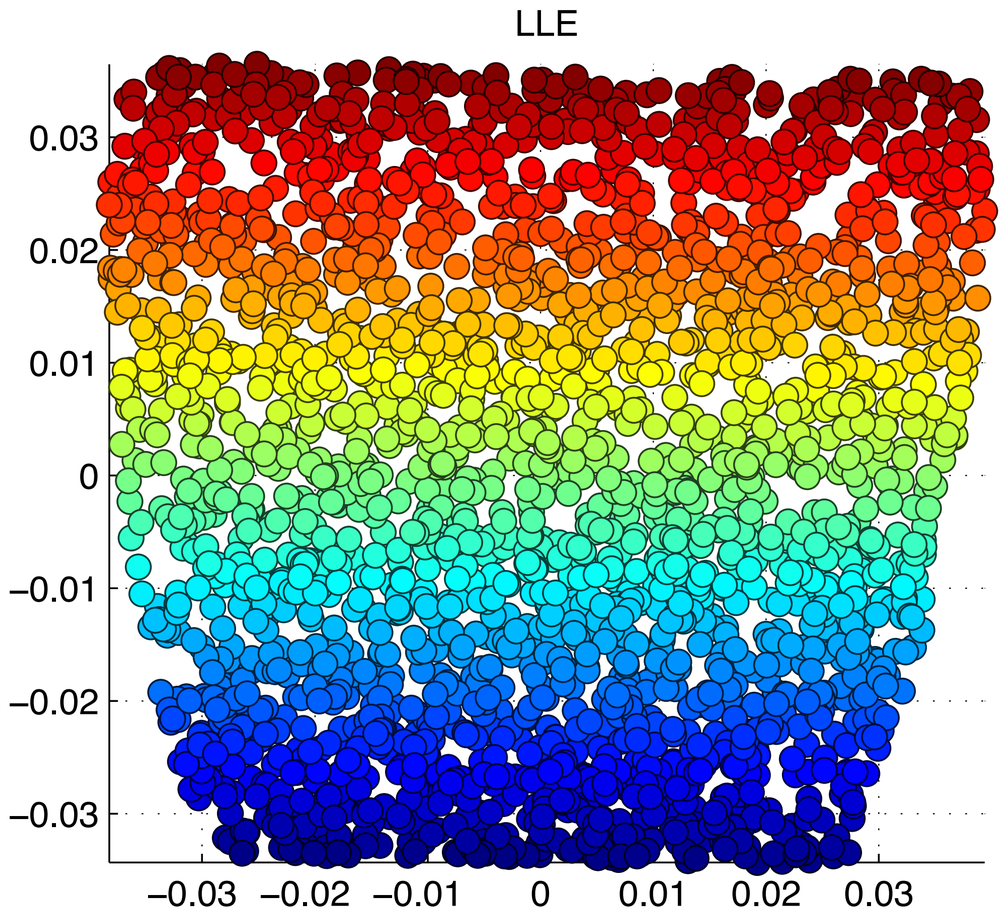
How to

- 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 $\mathbf{y}$

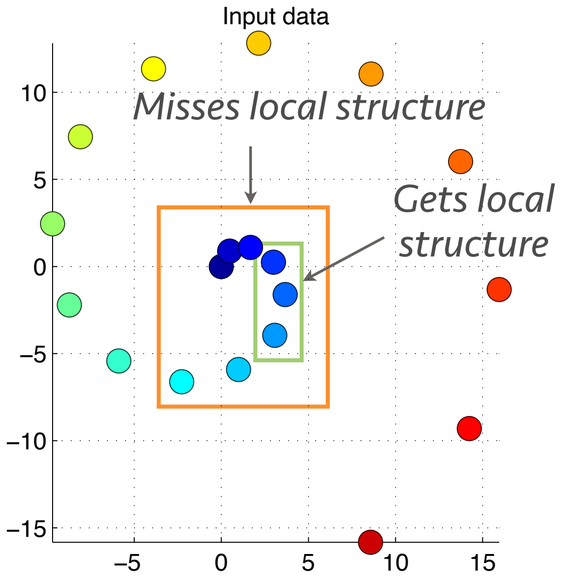
Swiss roll example

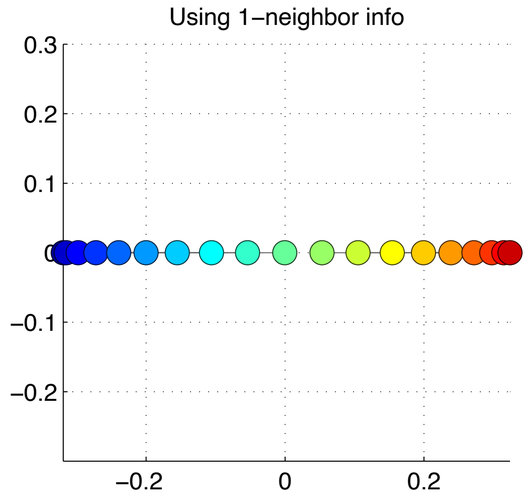




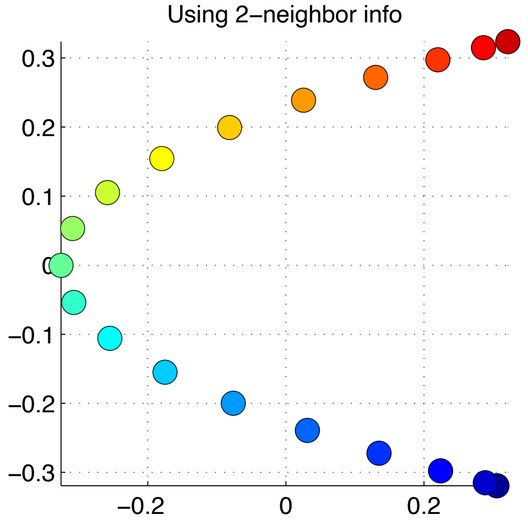
A note on manifold methods

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

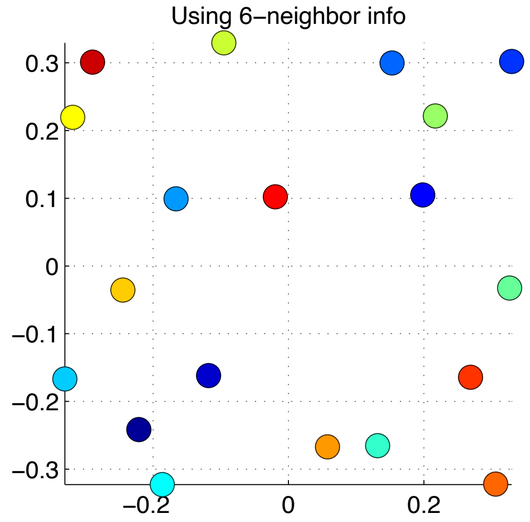




Sees the manifold as a line



Sees the manifold as a curve



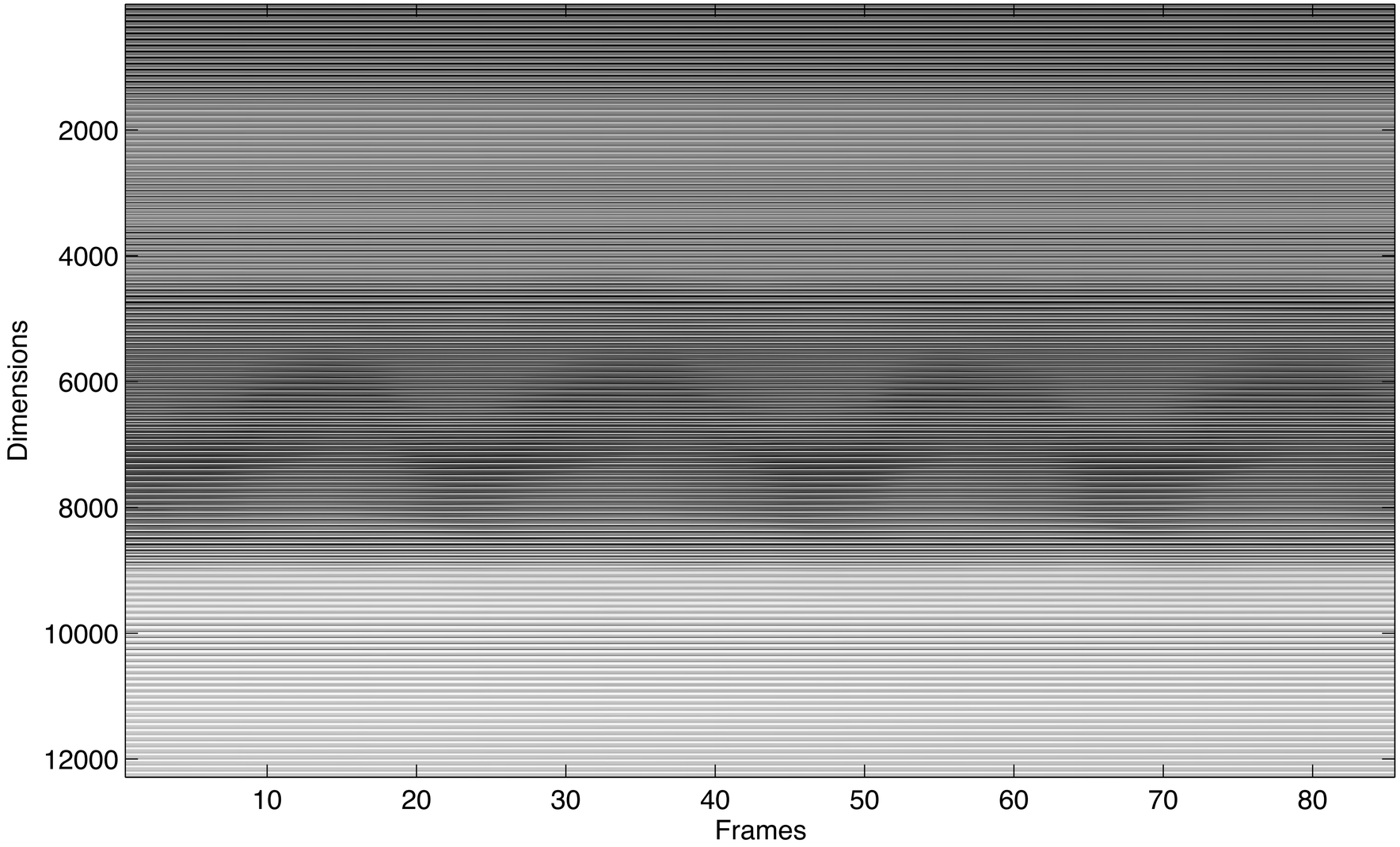
Sees the manifold as a mess!

A video example

- A high dimensional input
  - 128 x 96 pixels = 12,288 dimensions
- Low dimensional structure
  - Moving lips around
- Can we simplify the data?

Each frame is a pose

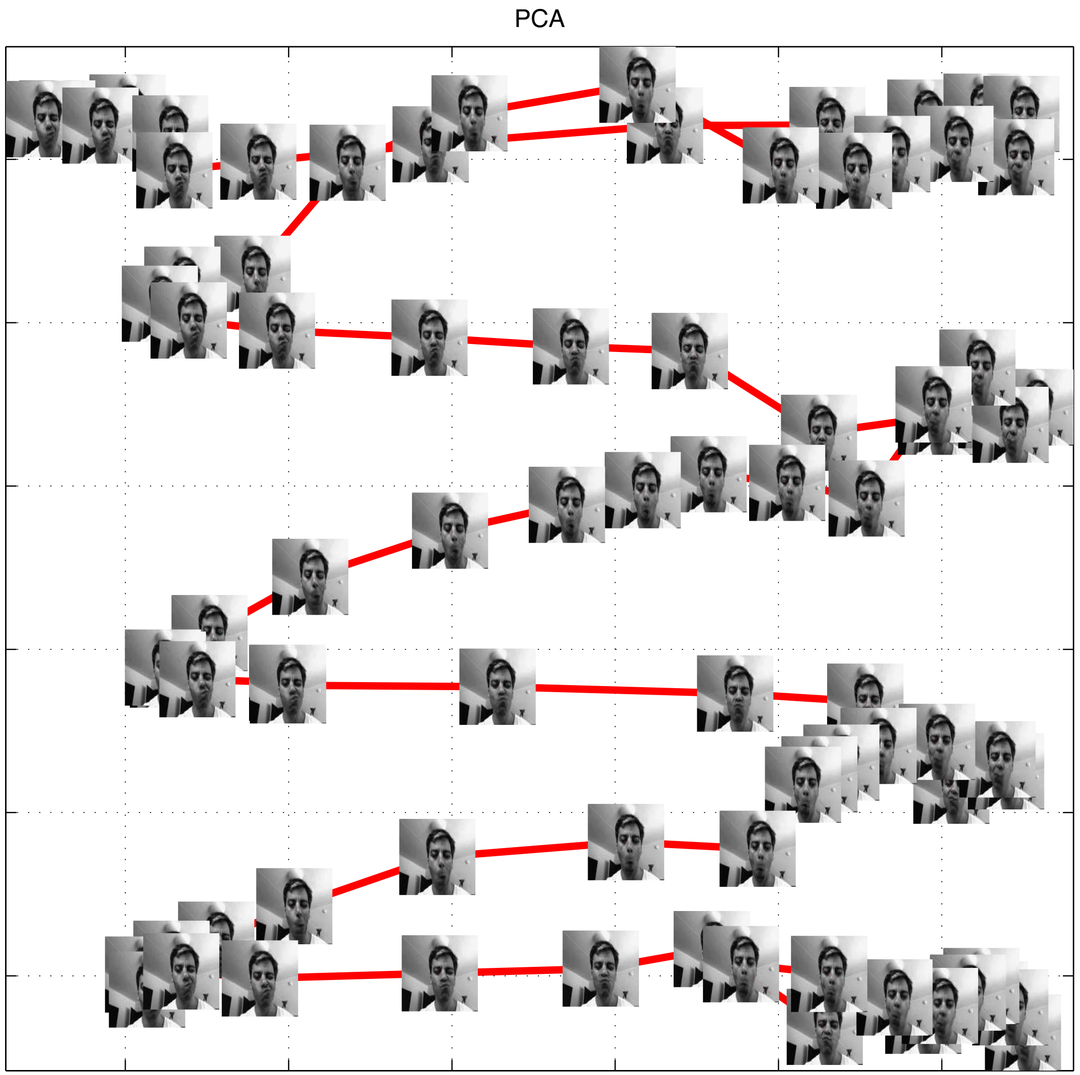
When unraveled



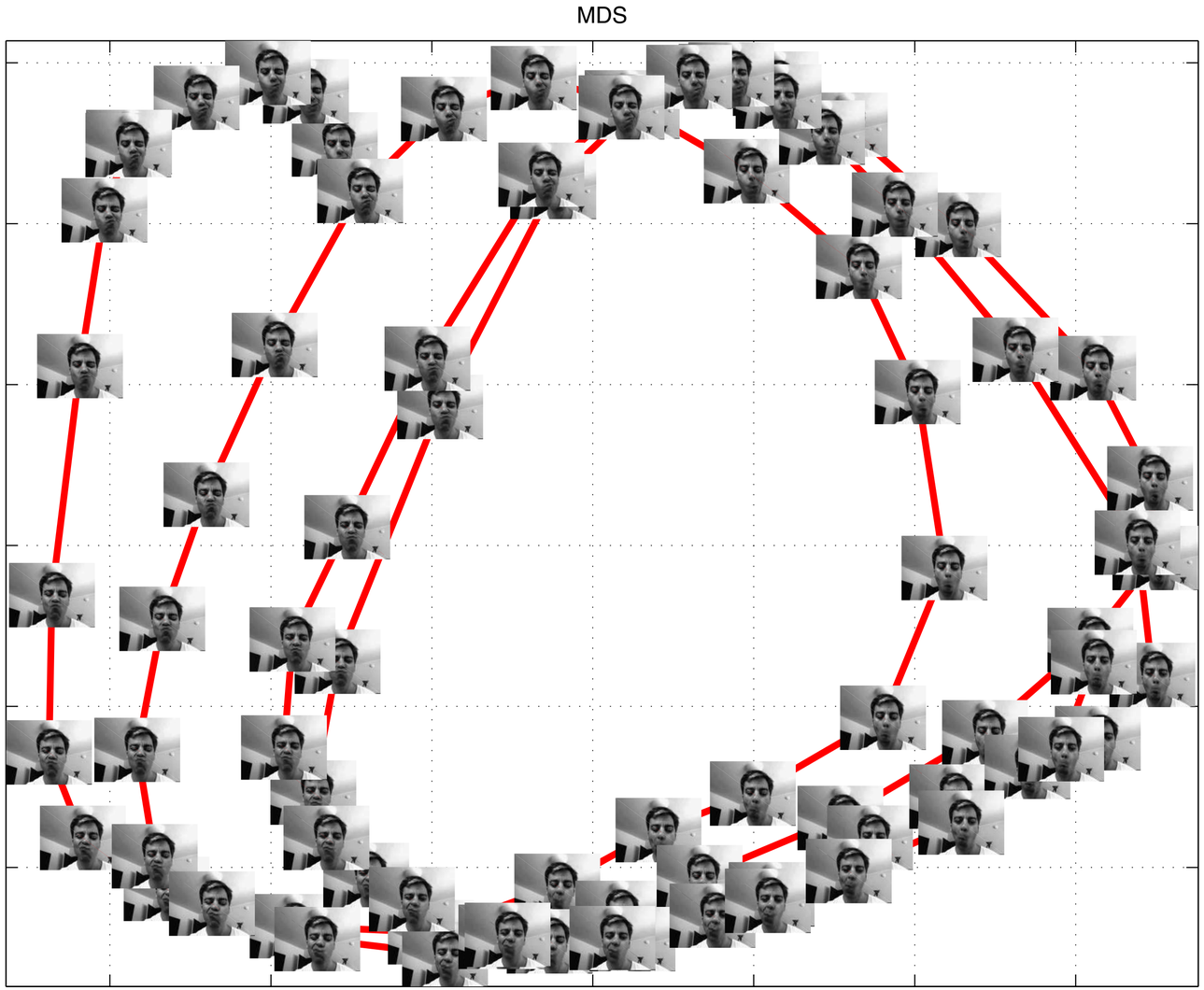
The manifold

- We only have a couple of dimensions
  - Primarily the movement of the lips
- The actual space is significantly lower dimensional than the video data
  - It is smooth
  - And it is “circular”

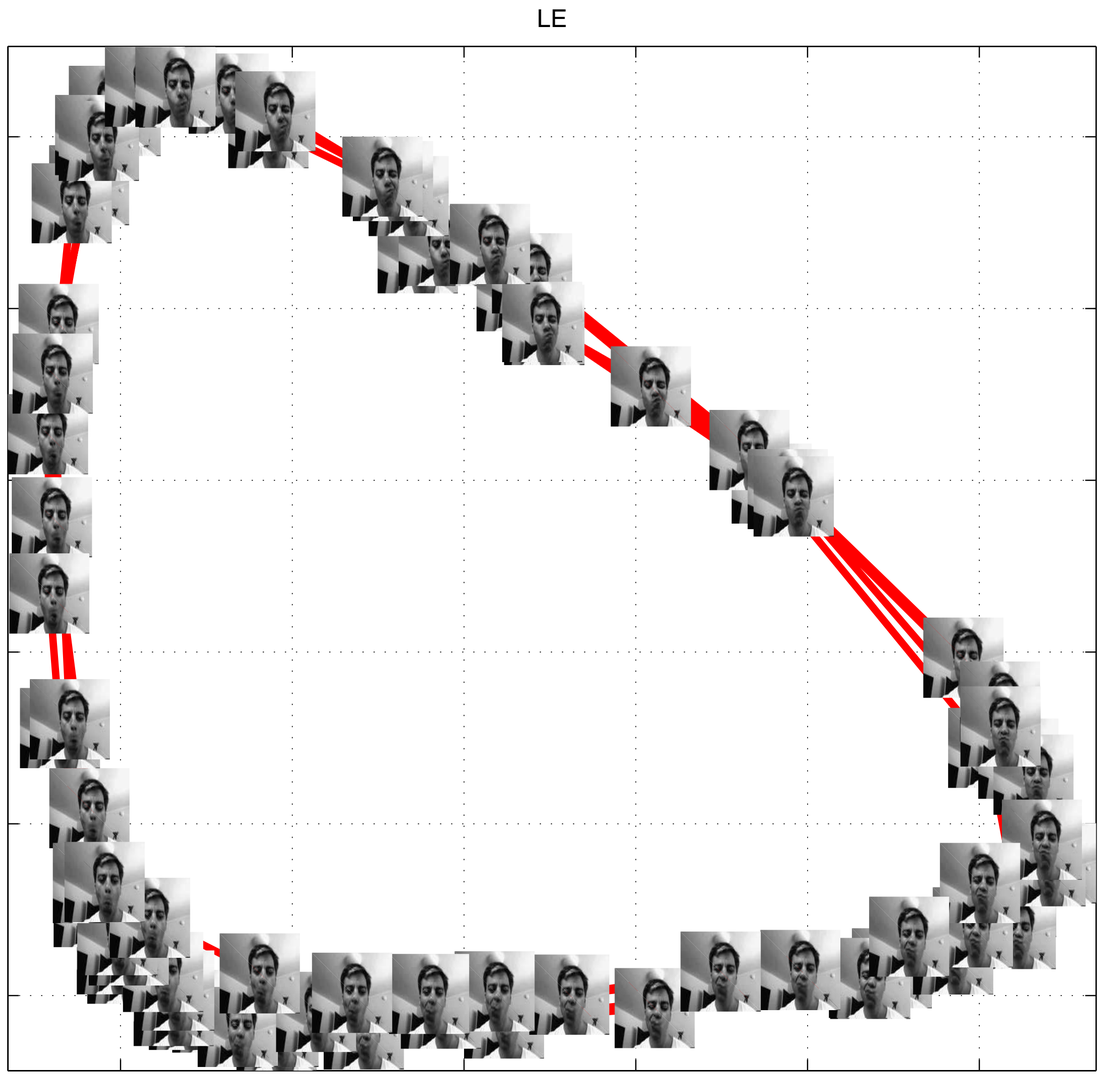
PCA



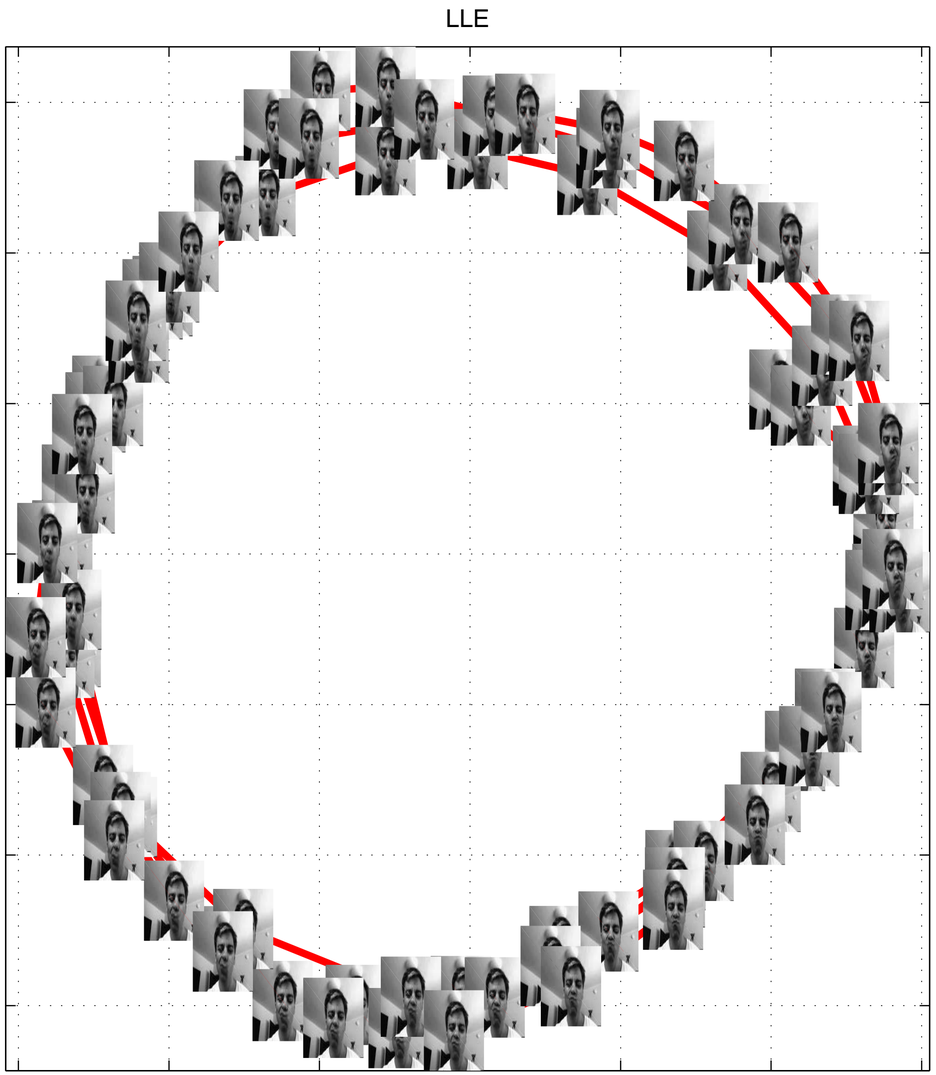
MDS



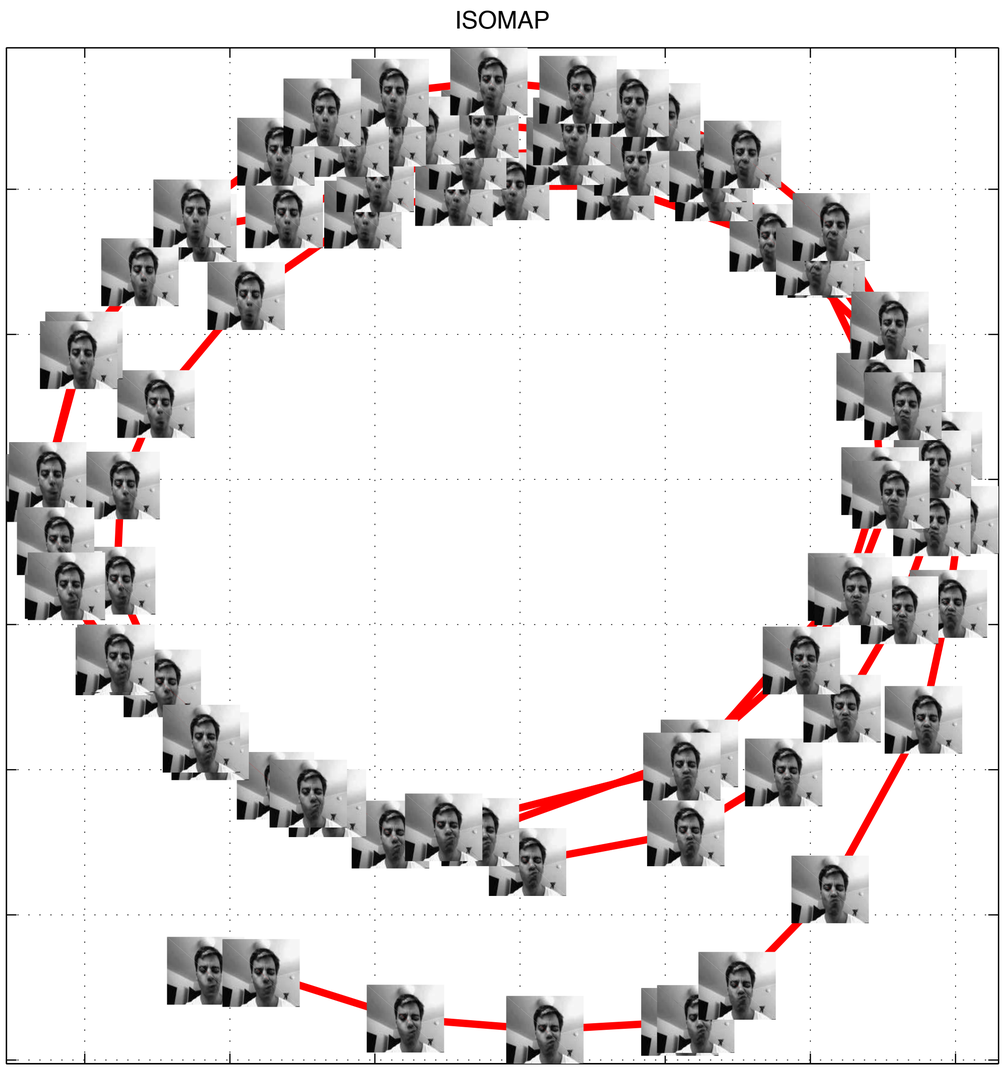
Laplacian Eigenmaps



LLE



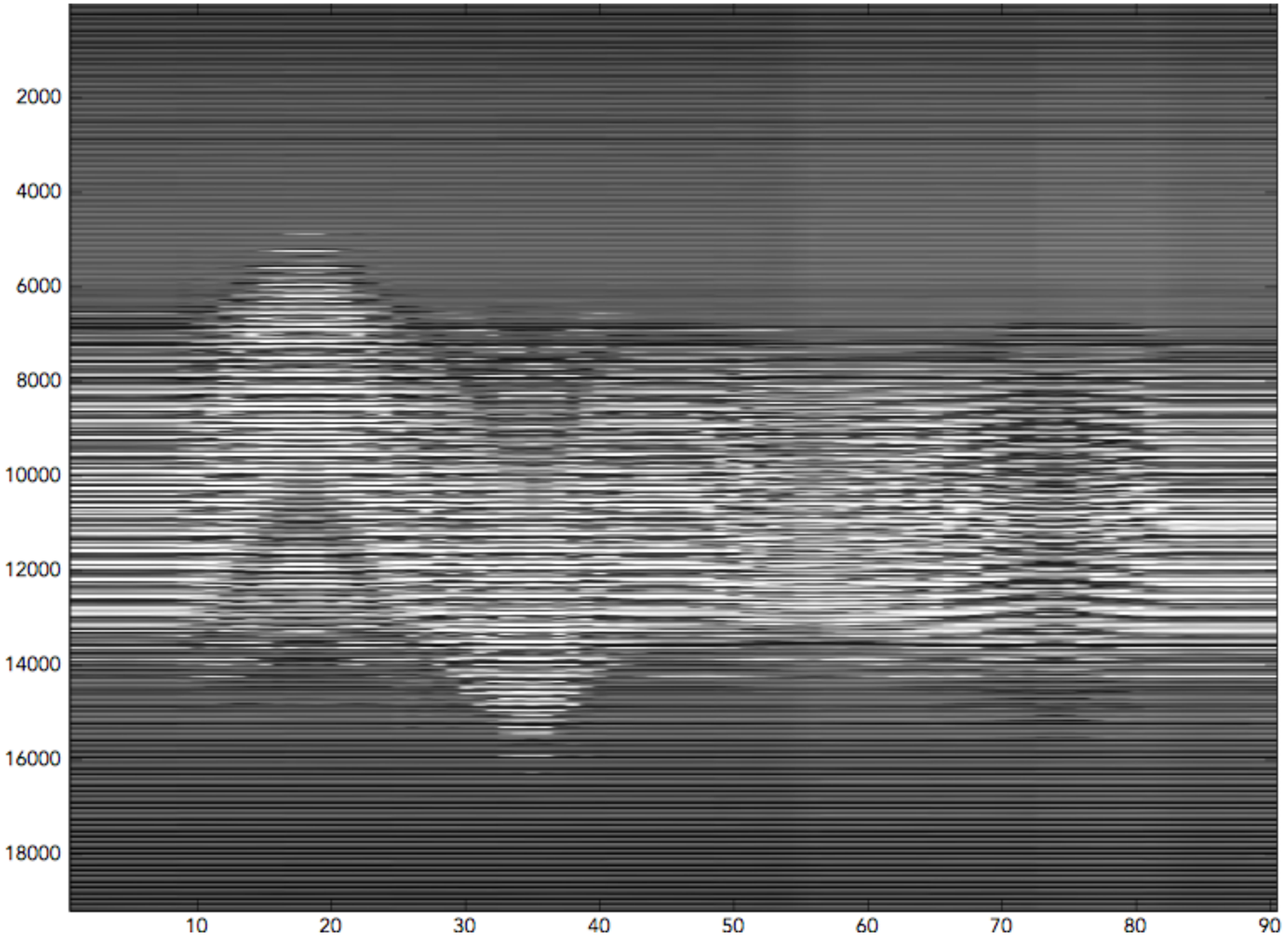
ISOMAP



Another example movie

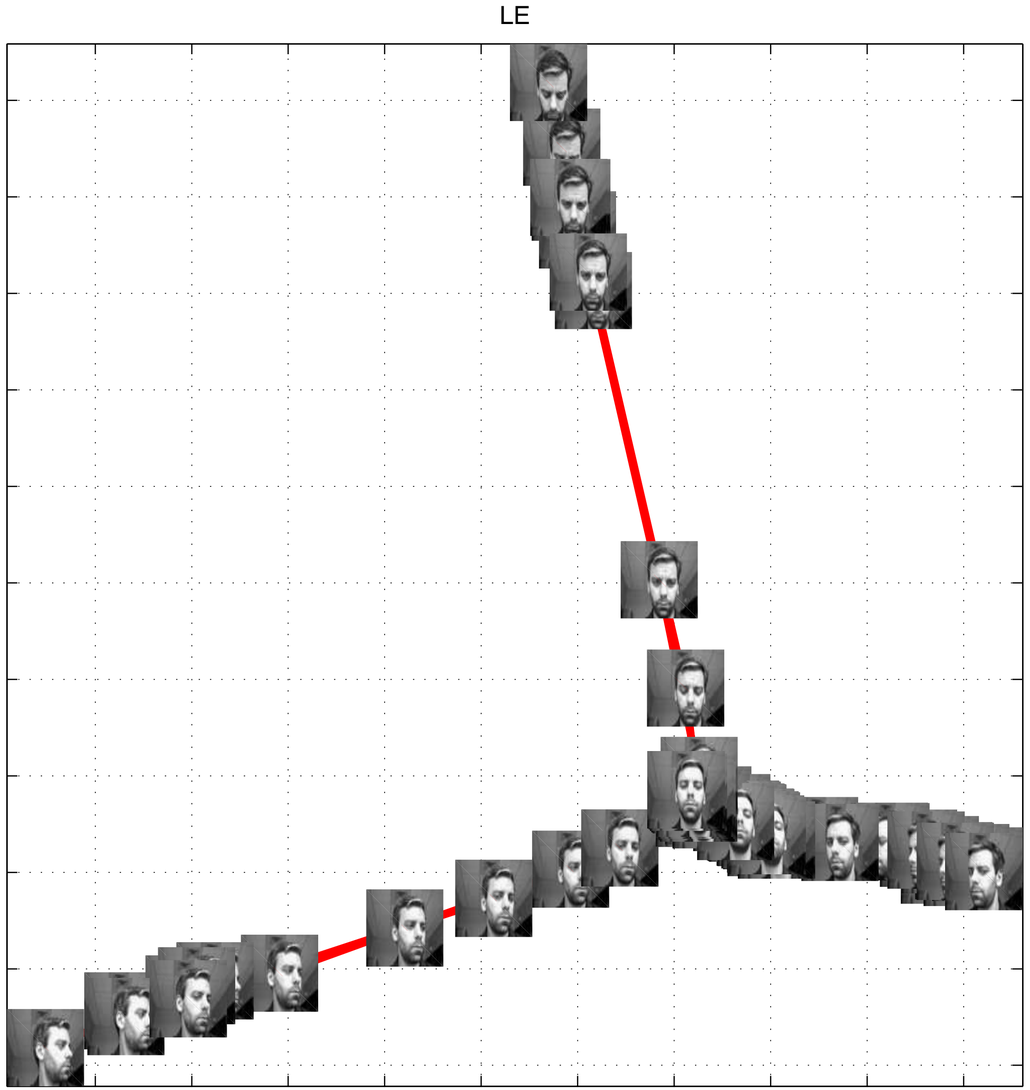
- This time we have distinct axes
  - Up-Down-Left-Right head movement

Input as a matrix



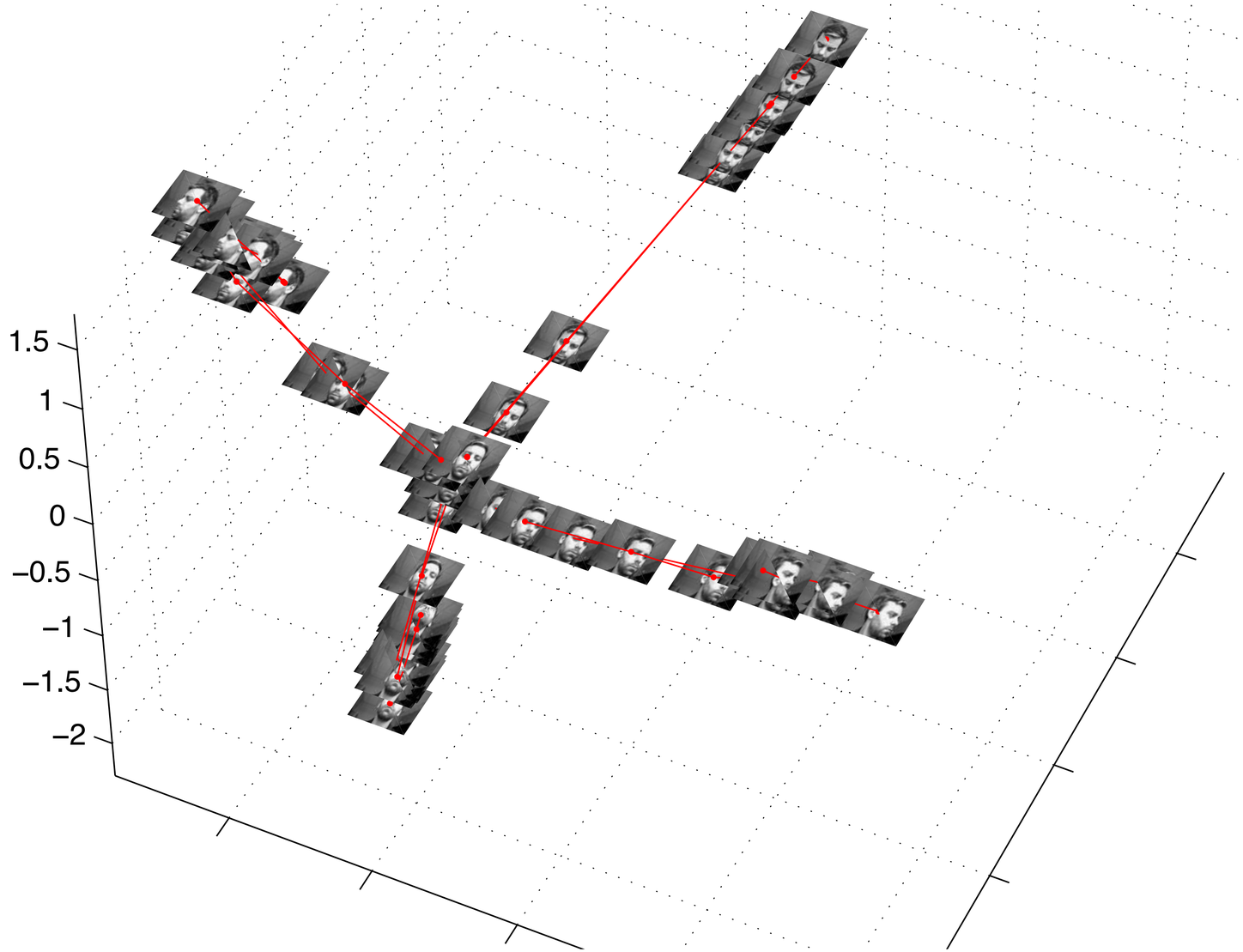
Laplacian Eigenmaps 2D

- 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