



CS545 – Machine Learning for Signal Processing

Dimensionality Reduction - Non-linear Approaches

11 September 2023

Today's lecture

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

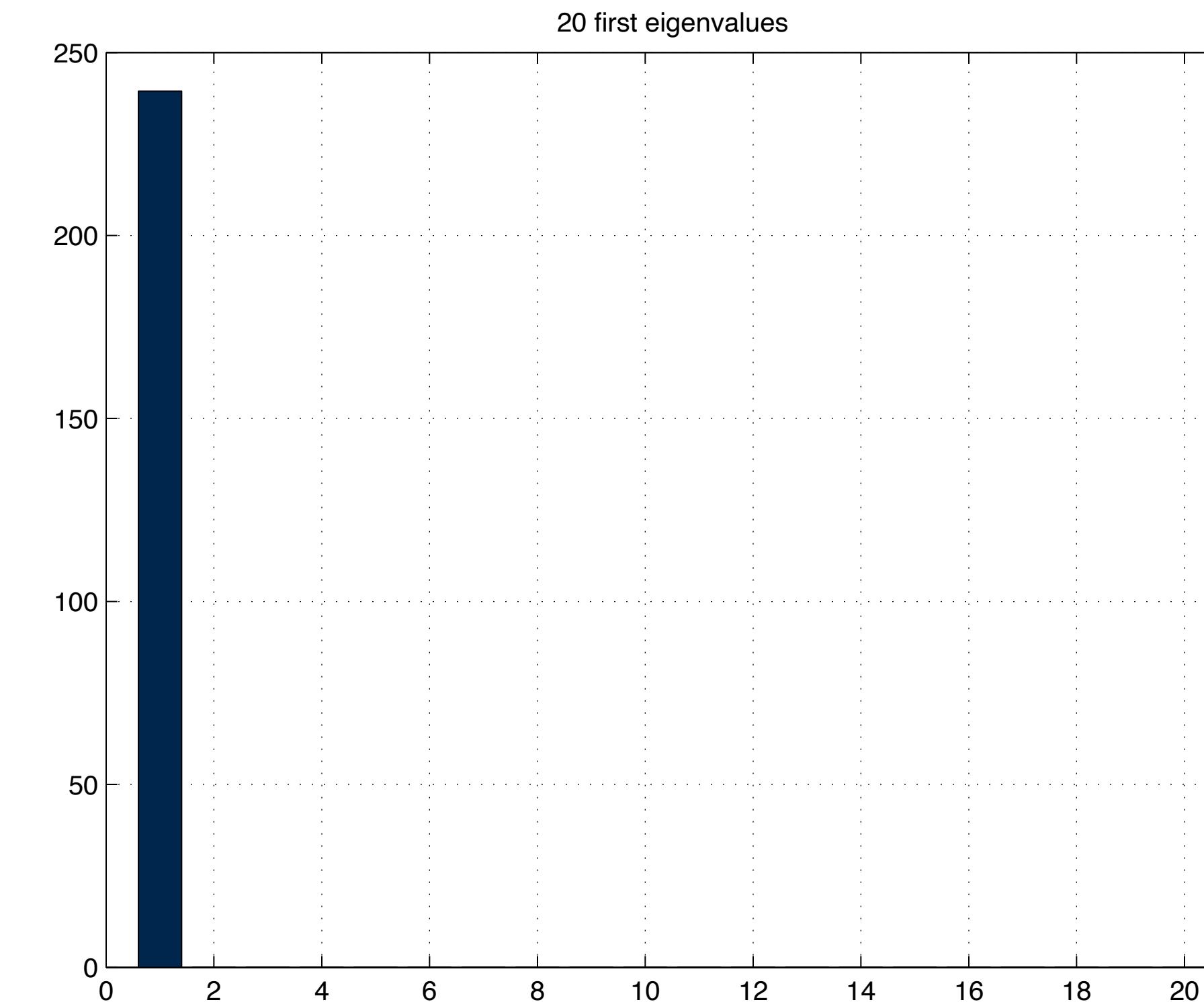
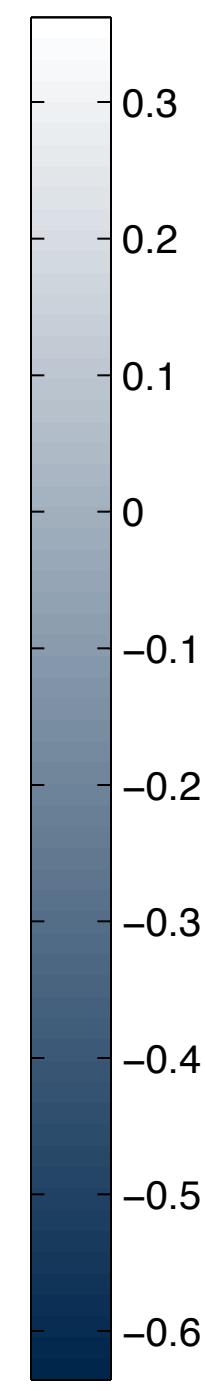
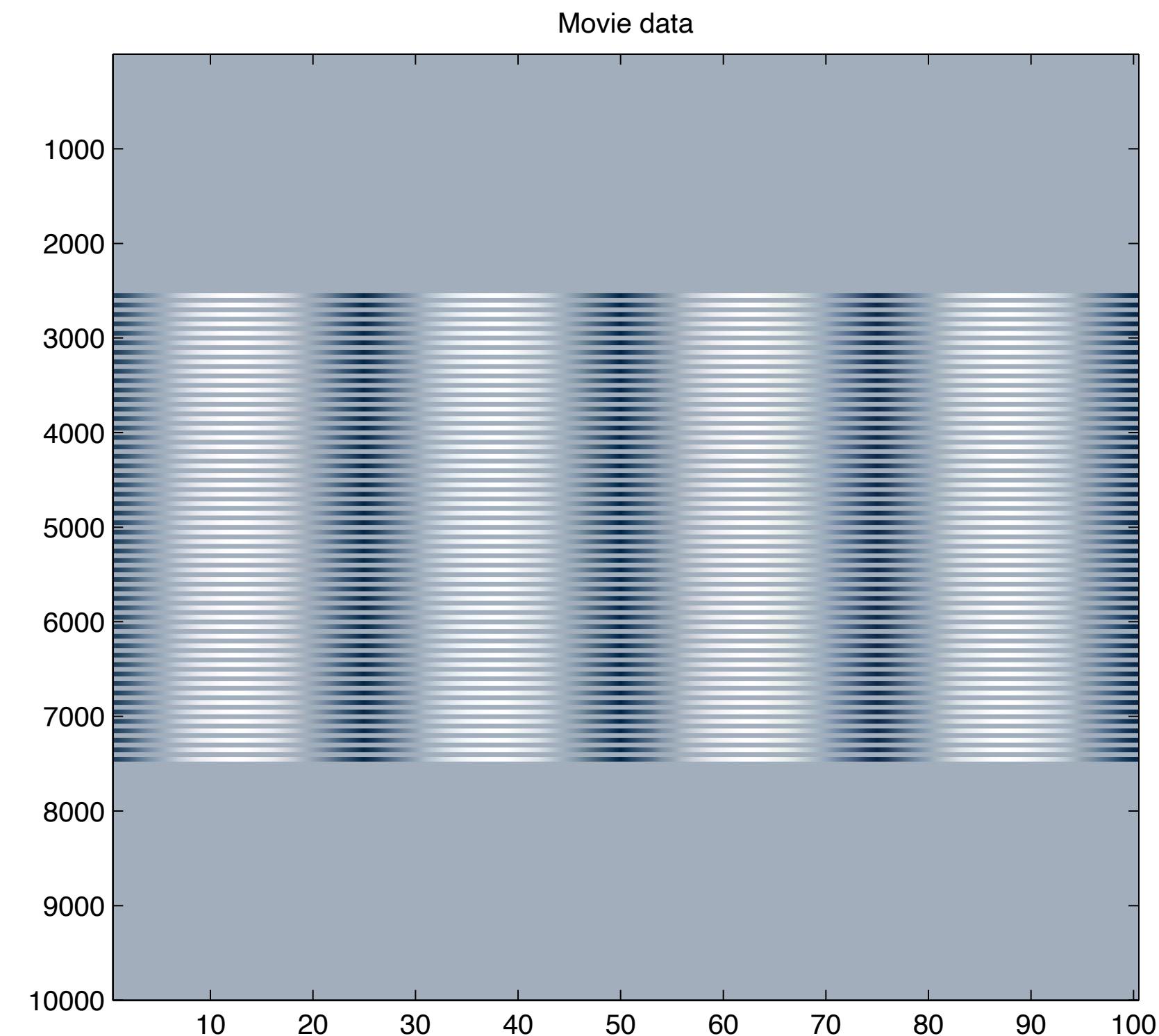
Why dimensionality reduction

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



Very low dimensionality!

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



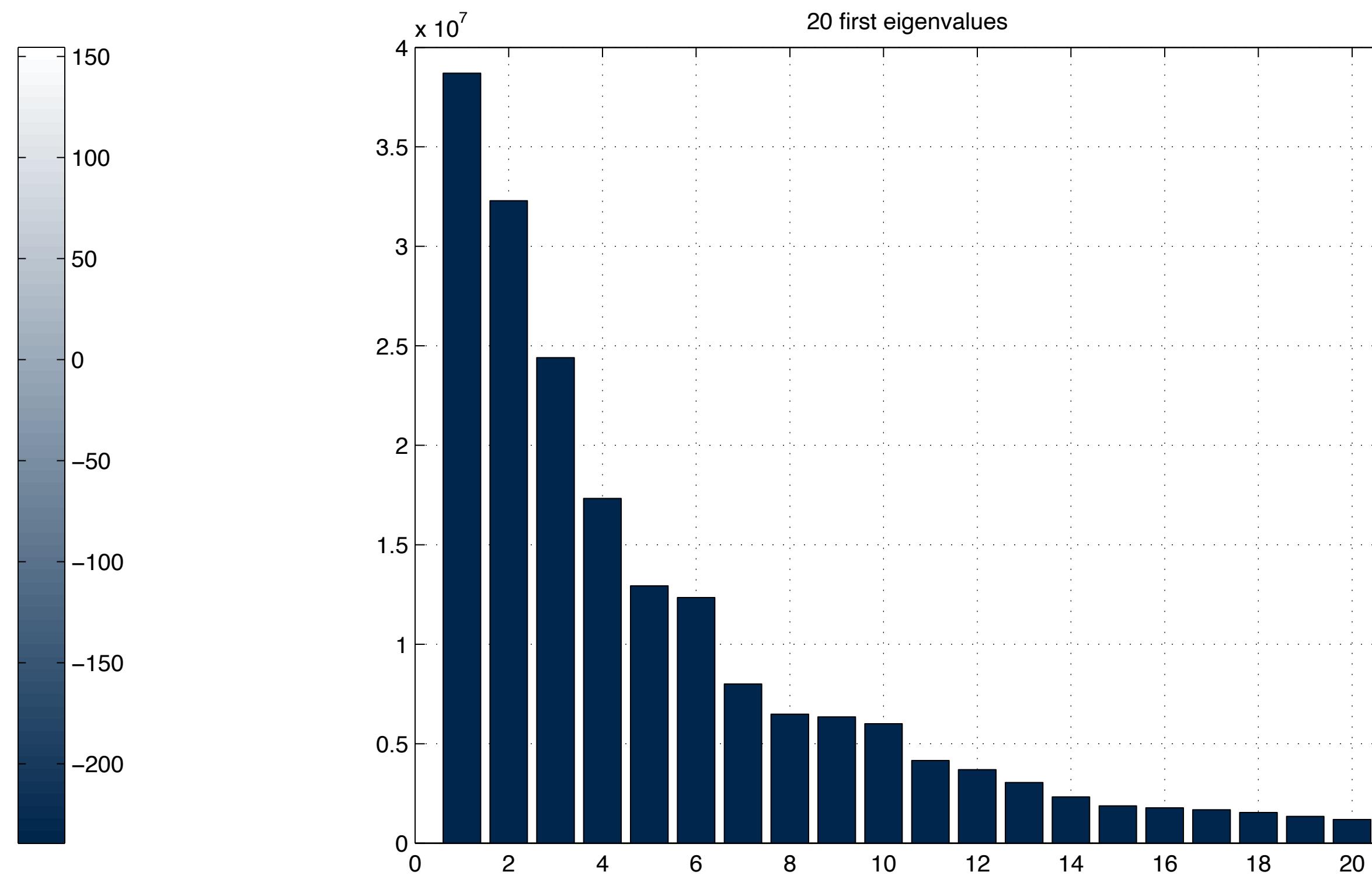
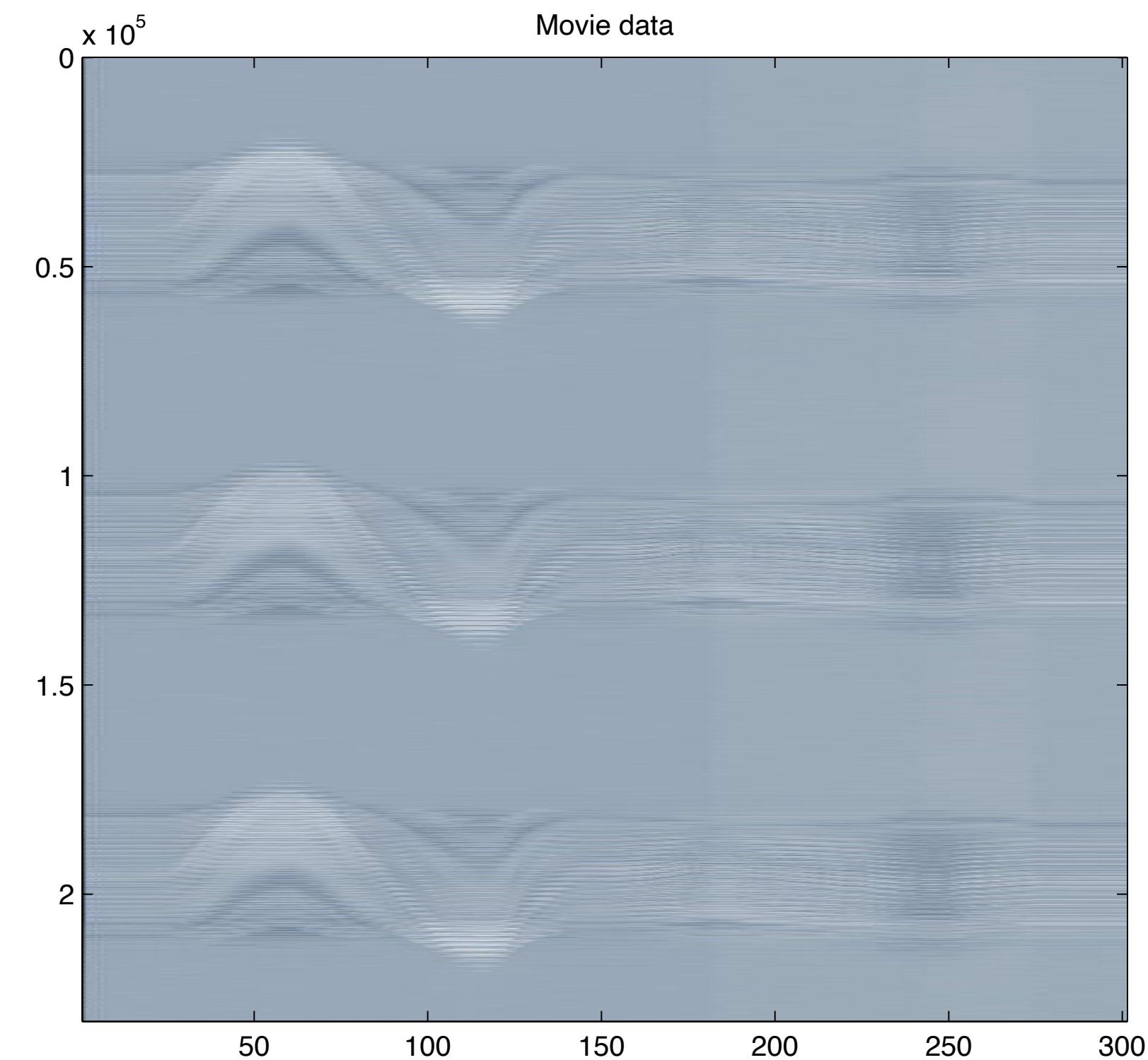
In real life though ...

- 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

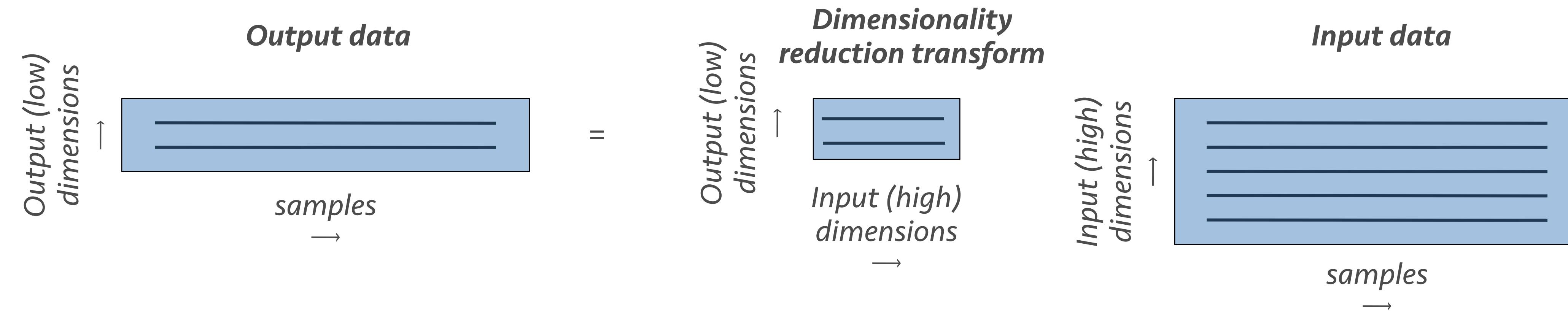


Linear dimensionality reduction

- Linear transform to drop dimensions

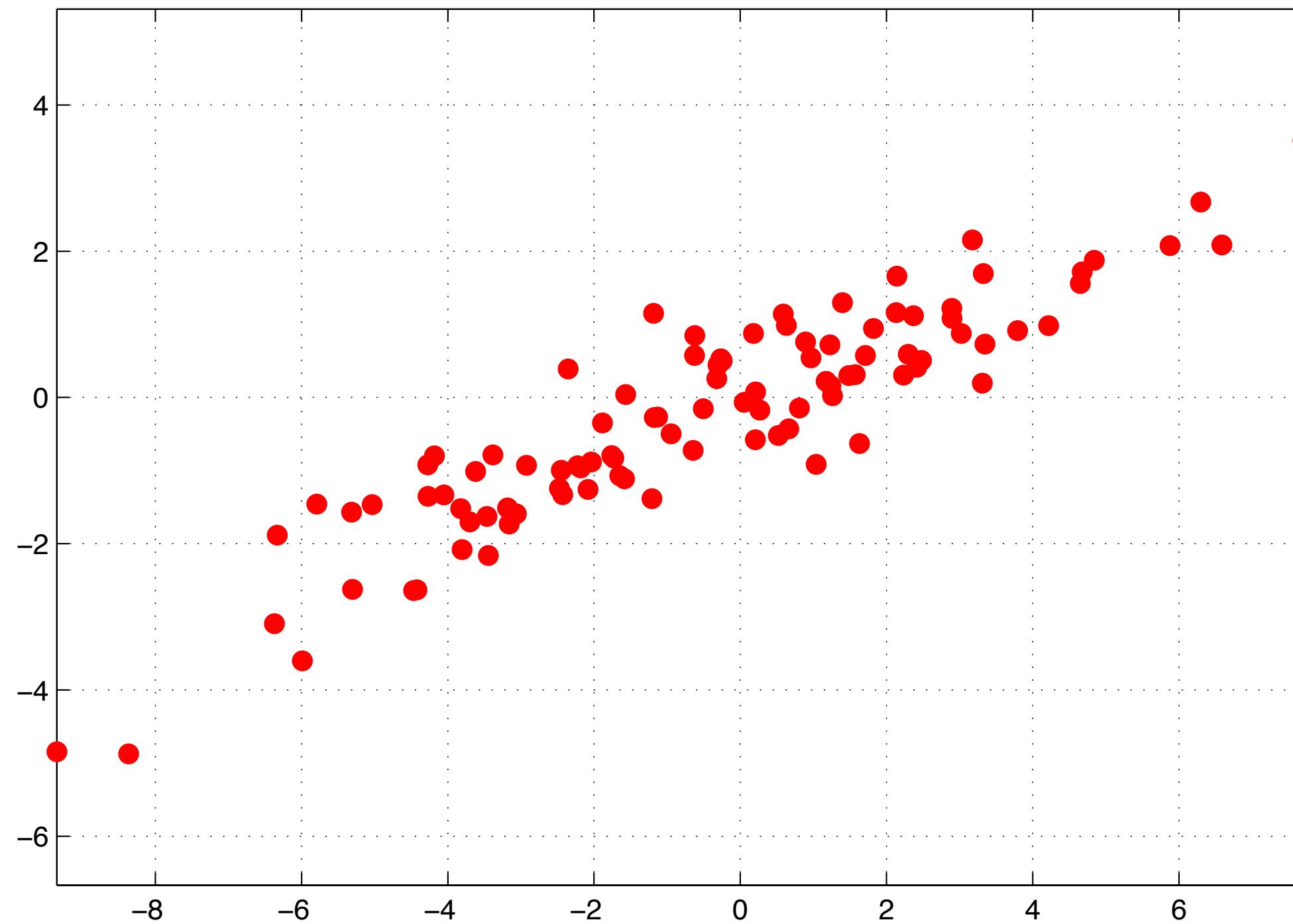
$$\mathbf{Y} = \mathbf{W} \cdot \mathbf{X}$$

- We saw this with PCA/NMF



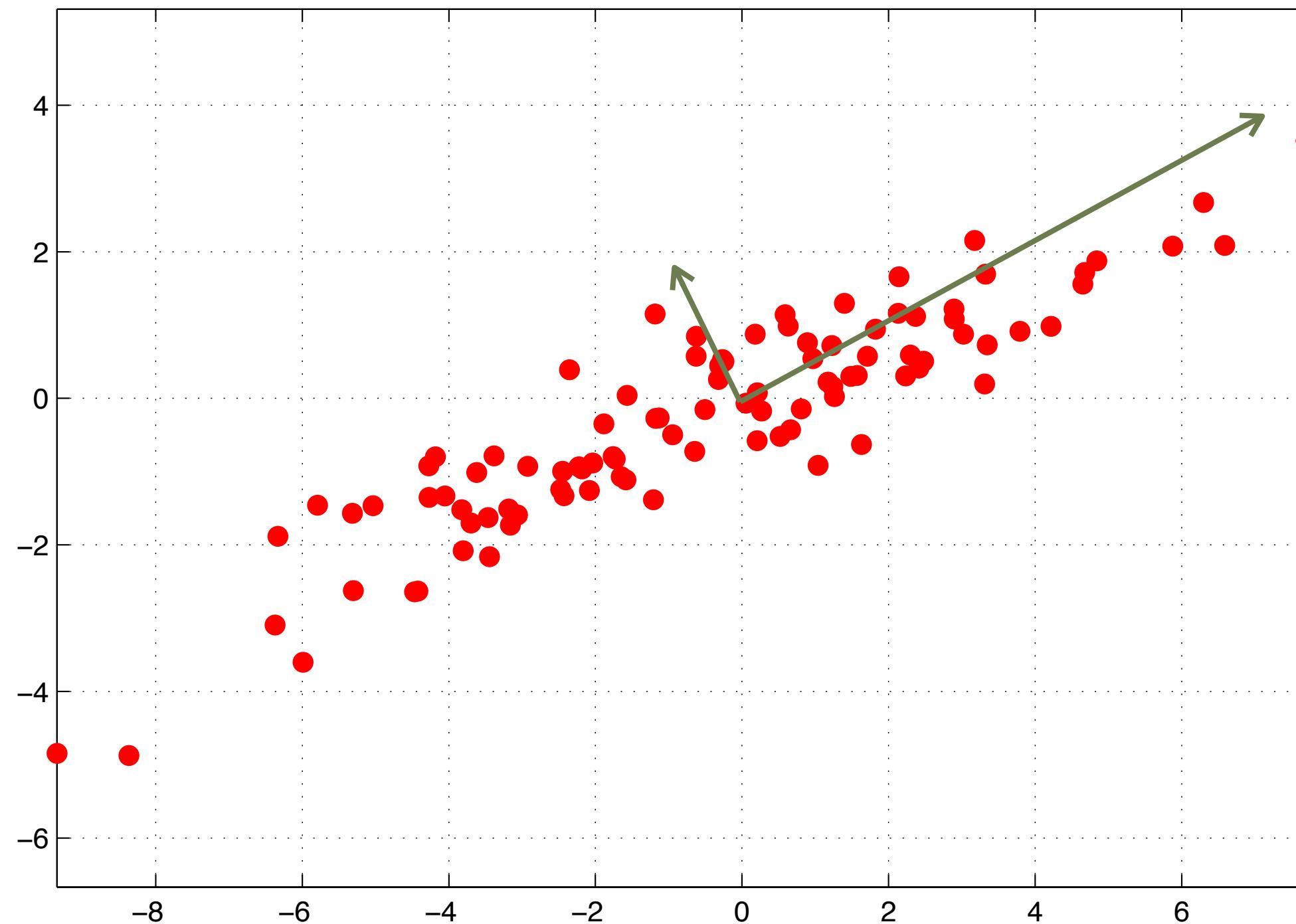
“Easy” data

- Linear transforms work fine with simple data



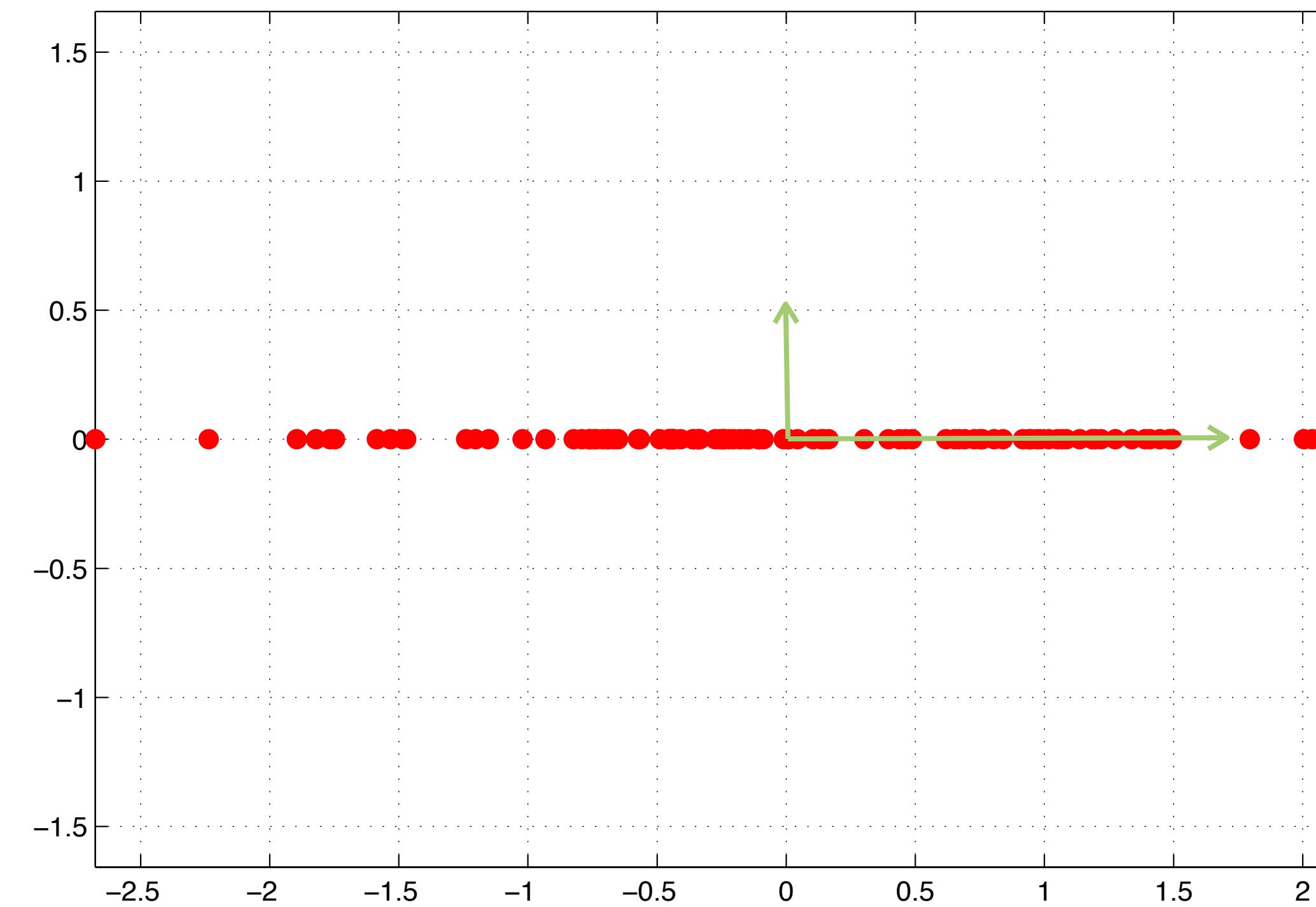
“Easy” data

- Principal components show directions of maximal variance



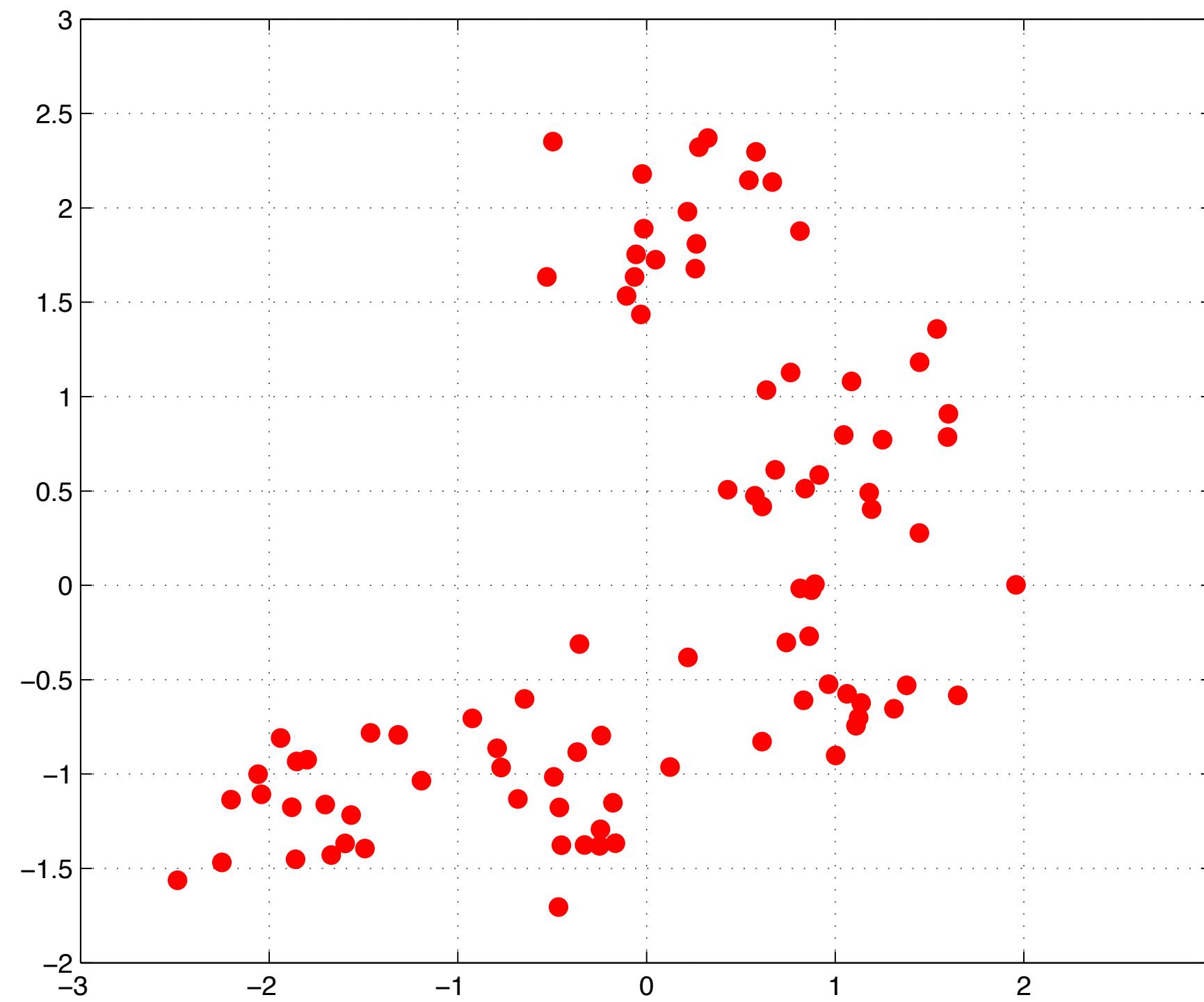
“Easy” data

- Keep only the significant dimensions to reduce dimensionality



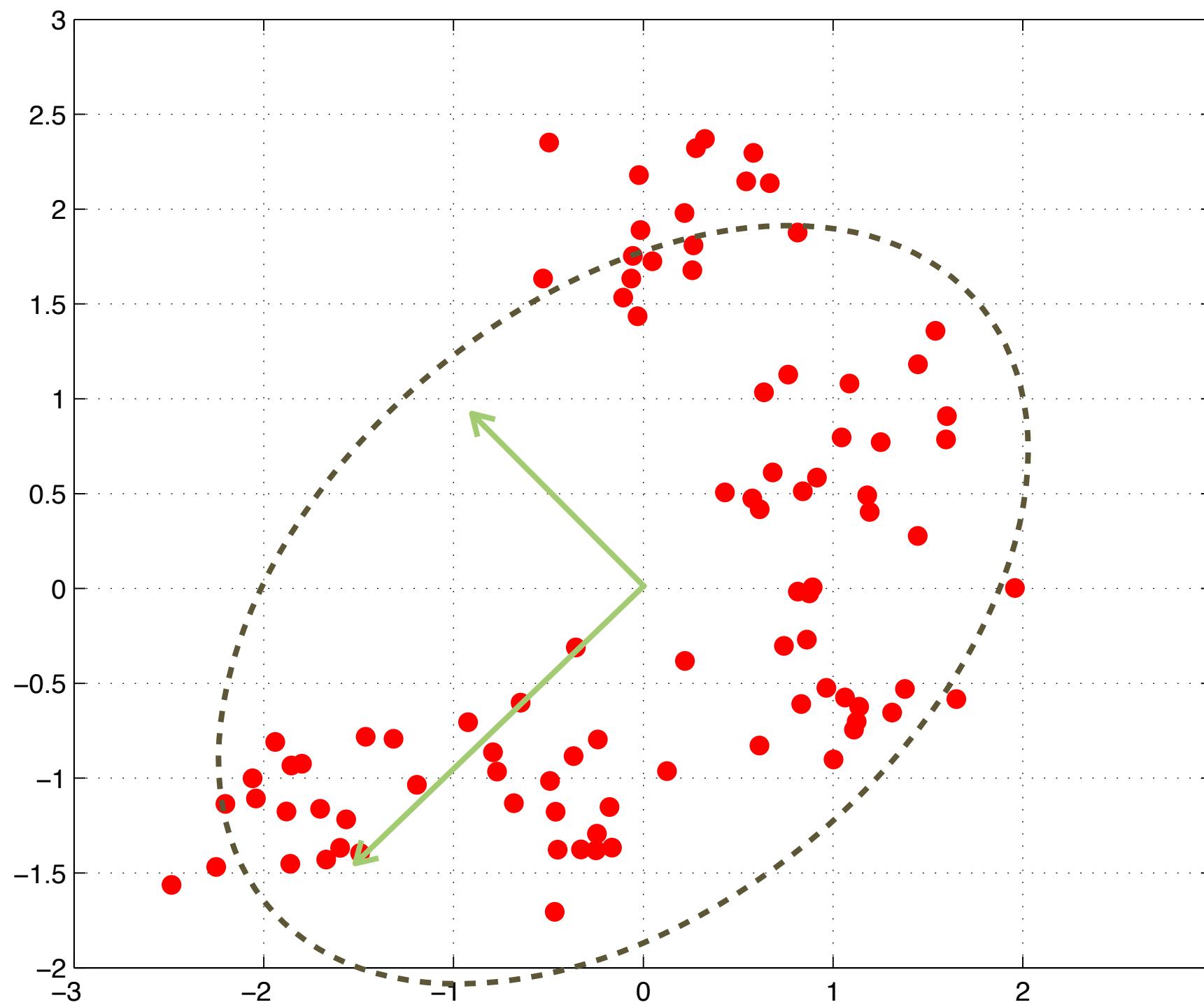
More complicated data

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



More complicated data

- Not much to interpret here ...

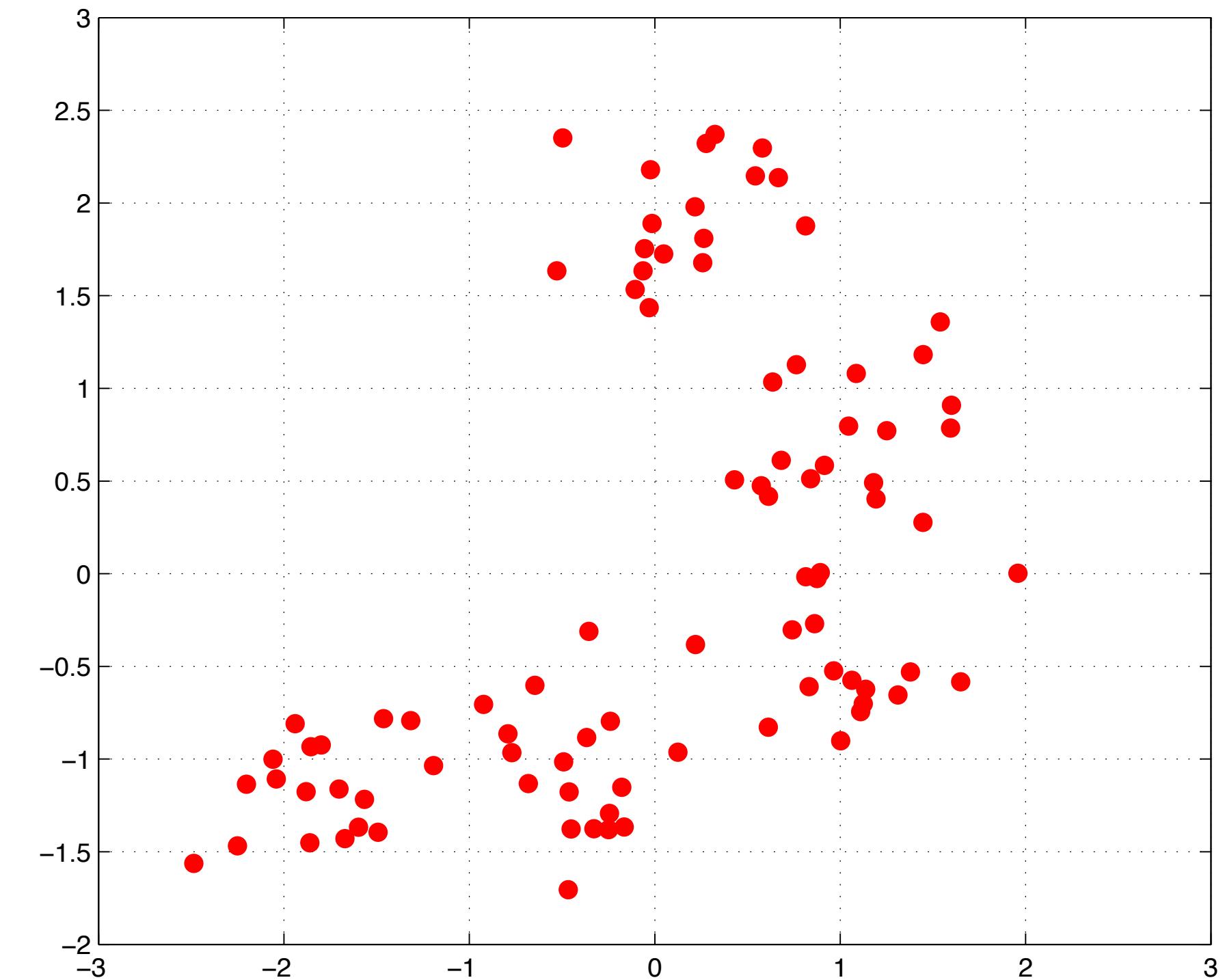


Pinpointing the problem

- The principal (and independent) 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

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

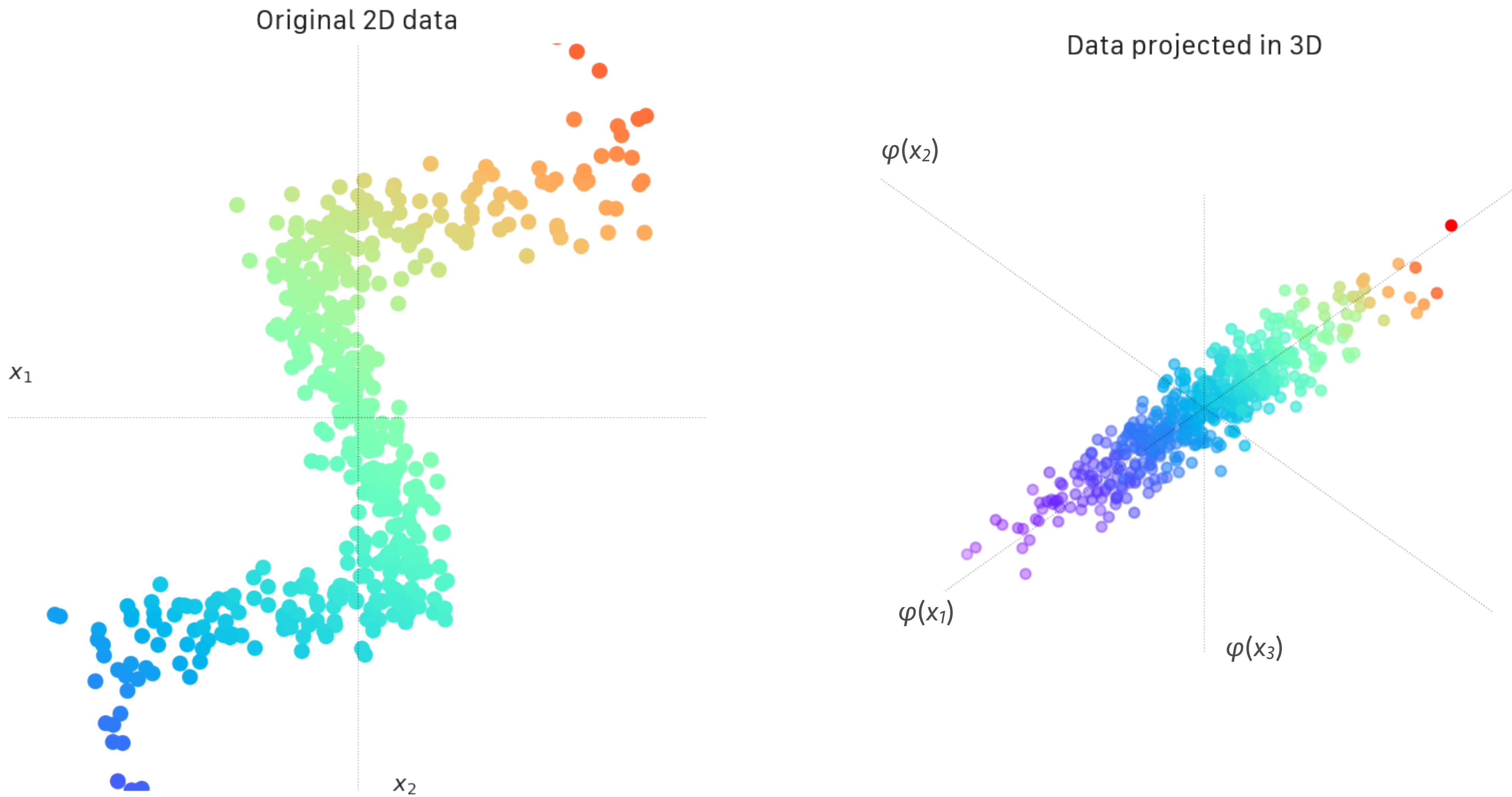
- Assume an unspecified non-linear mapping

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

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

What we hope for

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

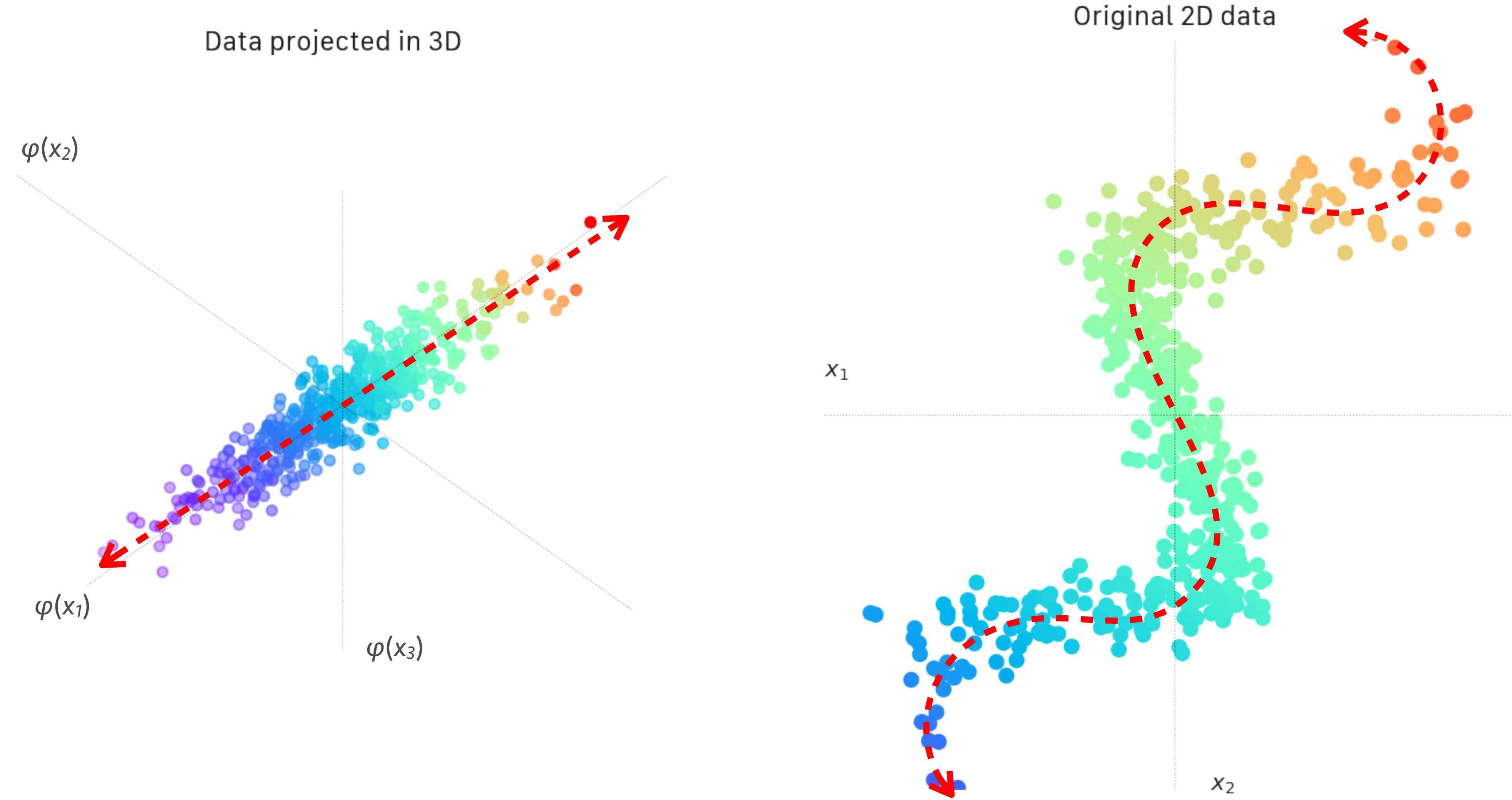


Now we can do PCA!

- 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:

$$\mathbf{S} = \text{Cov}(\mathbf{X}) \quad \leftarrow \text{Data covariance}$$

$$\mathbf{S} \cdot \mathbf{u}_i = \lambda_i \mathbf{u}_i \quad \leftarrow \text{Eigendecomposition}$$

- In kernel PCA we want to do:

$$\mathbf{C} = \text{Cov}(\phi(\mathbf{X})) \quad \leftarrow \text{Transformed data covariance}$$

$$\mathbf{C} \cdot \mathbf{v}_i = \lambda_i \mathbf{v}_i \quad \leftarrow \text{Eigendecomposition}$$

Some problems

- How do we choose the map φ ?
 - And will it actually do what we want?
- We are operating in more dimensions now!
 - 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 \mathbf{v}_i can be described by a linear combination of the input data, so we rewrite the above:

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

$$\mathbf{C} \cdot \mathbf{v}_i = \lambda \mathbf{v}_i \Rightarrow \frac{1}{N} \sum_n \phi(\mathbf{x}_n) \cdot \phi(\mathbf{x}_n)^\top \cdot \left[\sum_m a_{i,m} \phi(\mathbf{x}_m) \right] = \lambda_i \left[\sum_m a_{i,m} \phi(\mathbf{x}_m) \right]$$

The kernel trick

- Under certain constraints (more later)

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

- Where $K()$ is easier to compute, e.g.:

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

4-dimensional map

Rewriting the analysis

- The high-dim eigendecomposition was:

$$\mathbf{C} \cdot \mathbf{v}_i = \lambda \mathbf{v}_i \Rightarrow \frac{1}{N} \sum_n \phi(\mathbf{x}_n) \cdot \phi(\mathbf{x}_n)^\top \cdot \left[\sum_m a_{i,m} \phi(\mathbf{x}_m) \right] = \lambda_i \left[\sum_m a_{i,m} \phi(\mathbf{x}_m) \right]$$

- We can now rewrite as a lower-dim version:

- Multiply on both sides with $\phi(\mathbf{x}_l)$

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

- And over all l : $\Rightarrow \mathbf{K} \cdot \mathbf{K} \cdot \mathbf{a}_i = \lambda_i N \mathbf{K} \cdot \mathbf{a}_i$

That's solved easily

- It is an eigendecomposition once again!

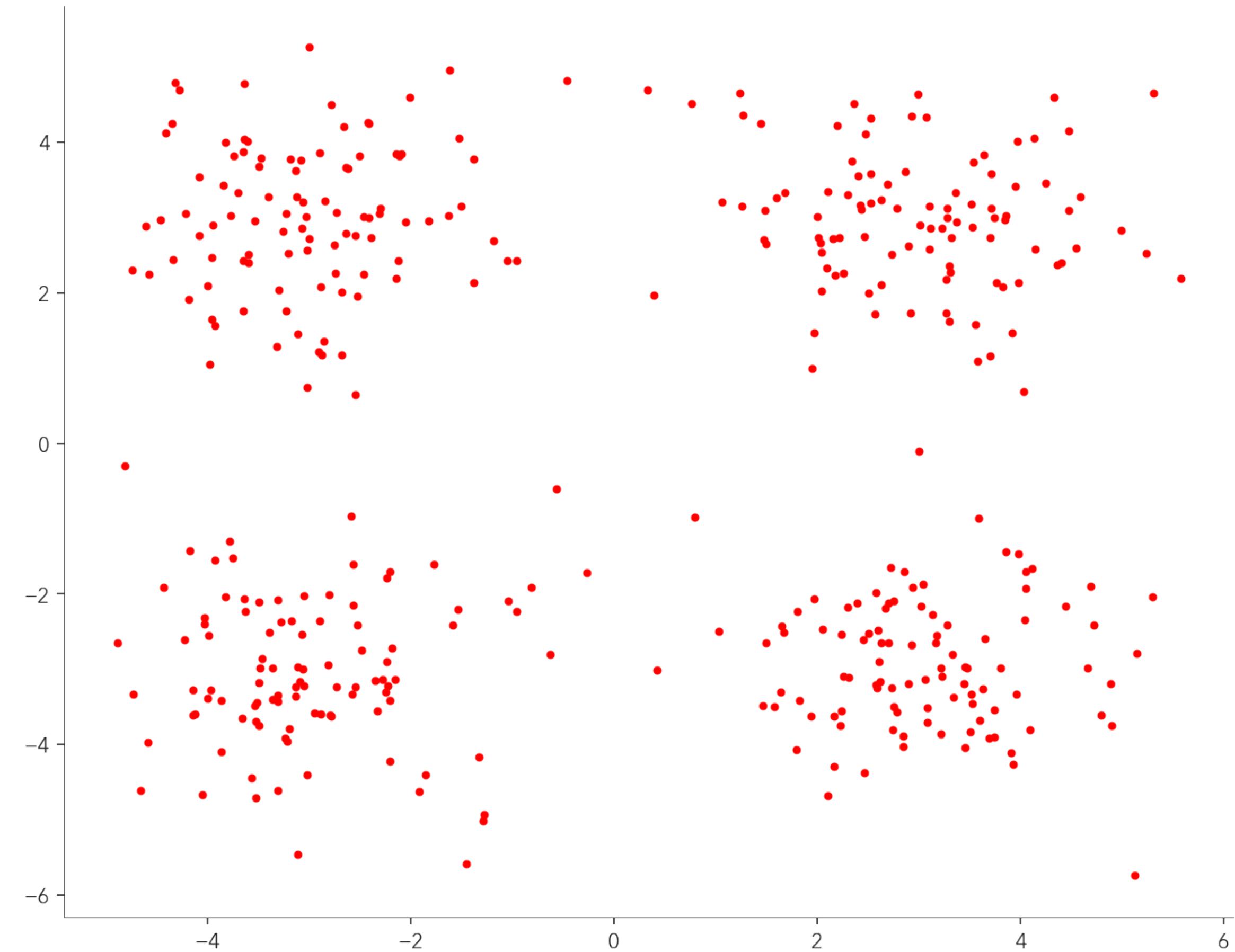
$$\mathbf{K} \cdot \mathbf{a}_i = \lambda_i N \mathbf{a}_i$$

- The non-linear principal components are \mathbf{a}_i
- The matrix \mathbf{K} is now $N \times N$ 

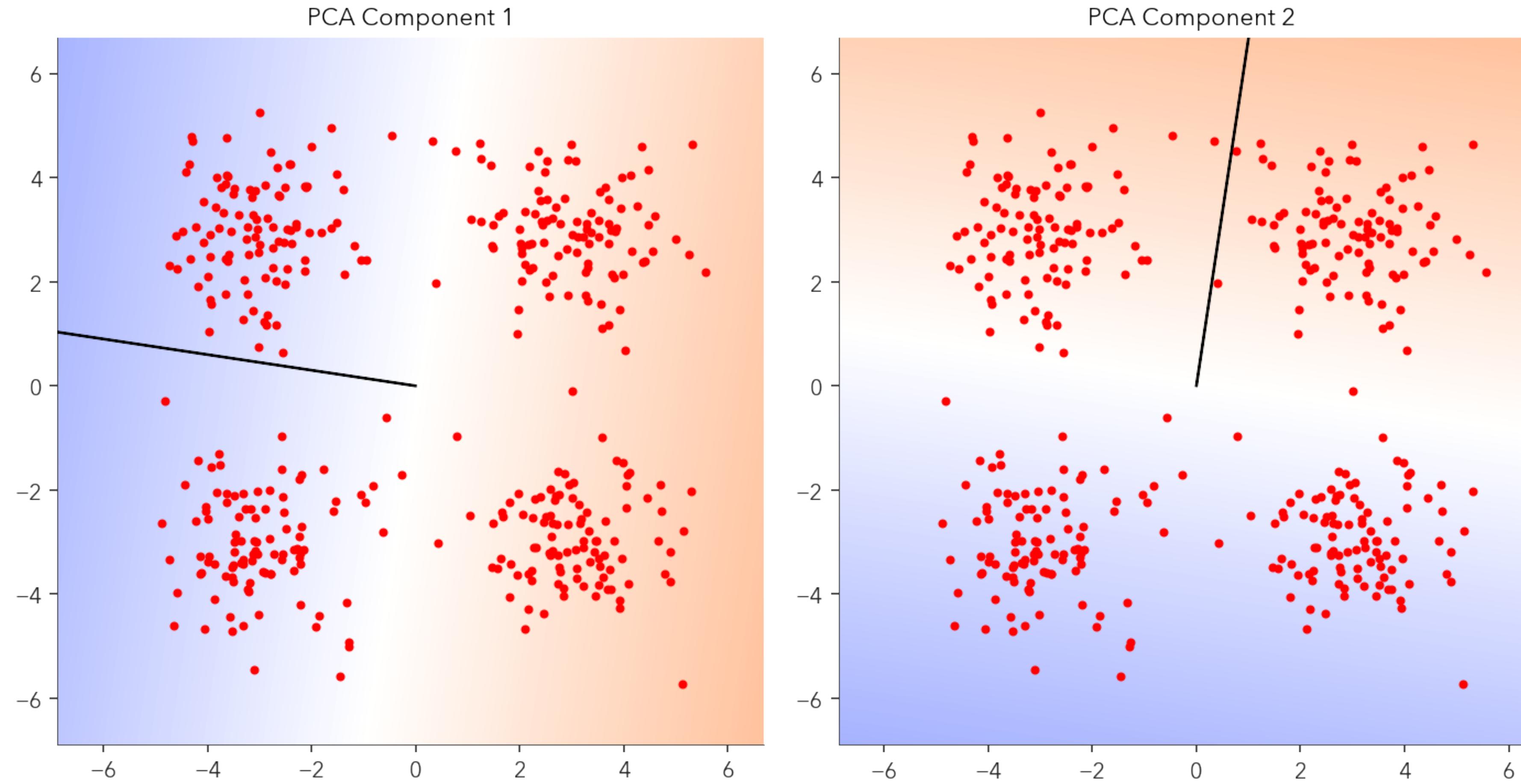
of samples, not dimensions like before!
- In practice we need to ensure that \mathbf{K} is also positive definite (i.e. $\phi(\mathbf{x})$ is zero mean)
 - Not hard to do, but we skip the details for now

An example

- Data is not good for PCA
 - Isn't Gaussian and has a multimodal distribution
- Use this kernel:
$$K(x,y) = e^{-\|x-y\|^2}$$
 - Eigenvectors will be in terms of some high-D space



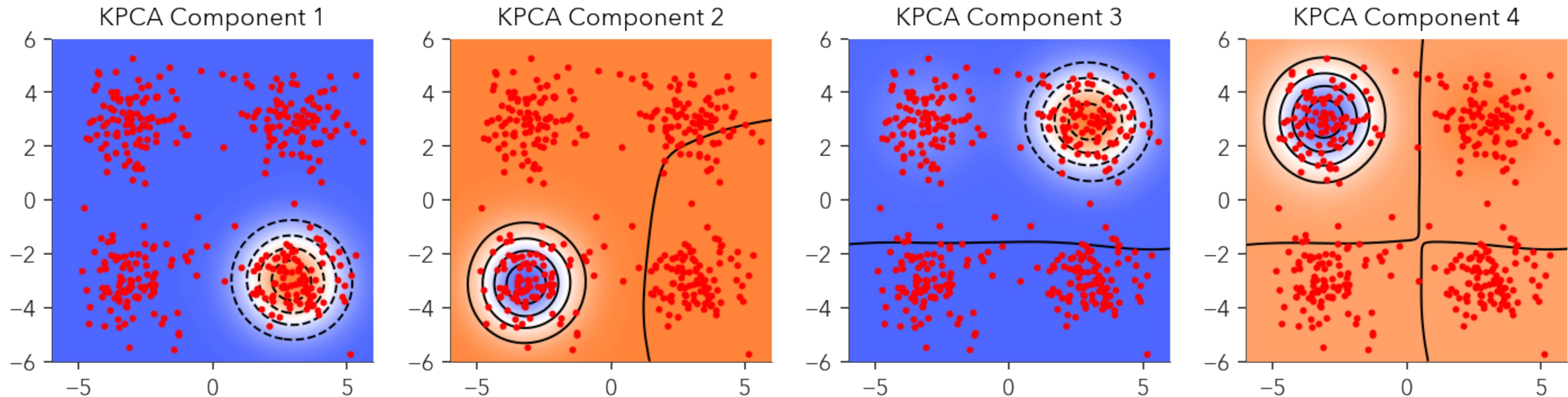
PCA components are not informative



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

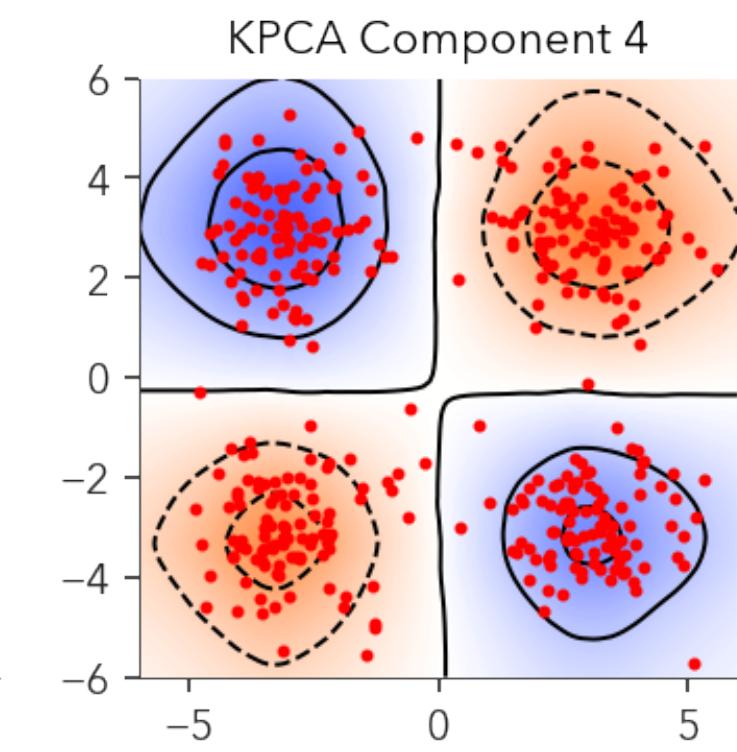
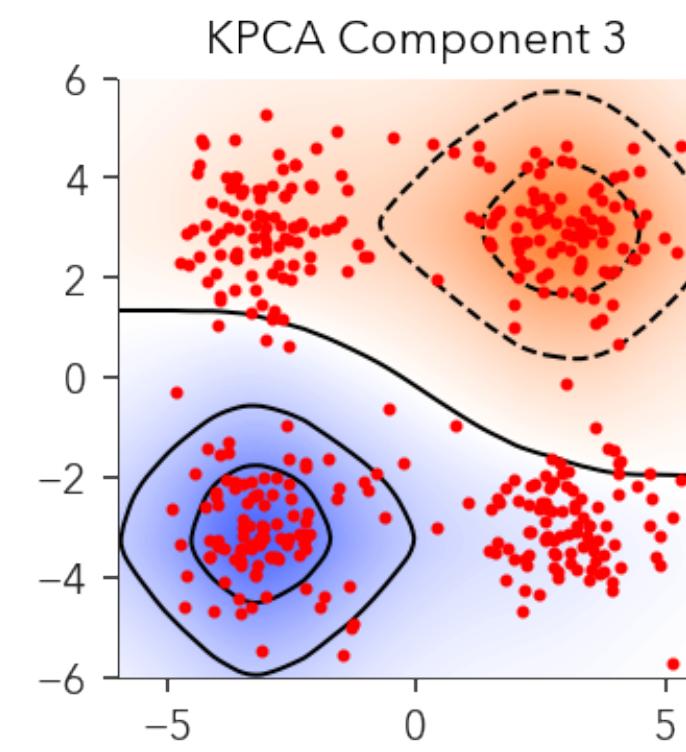
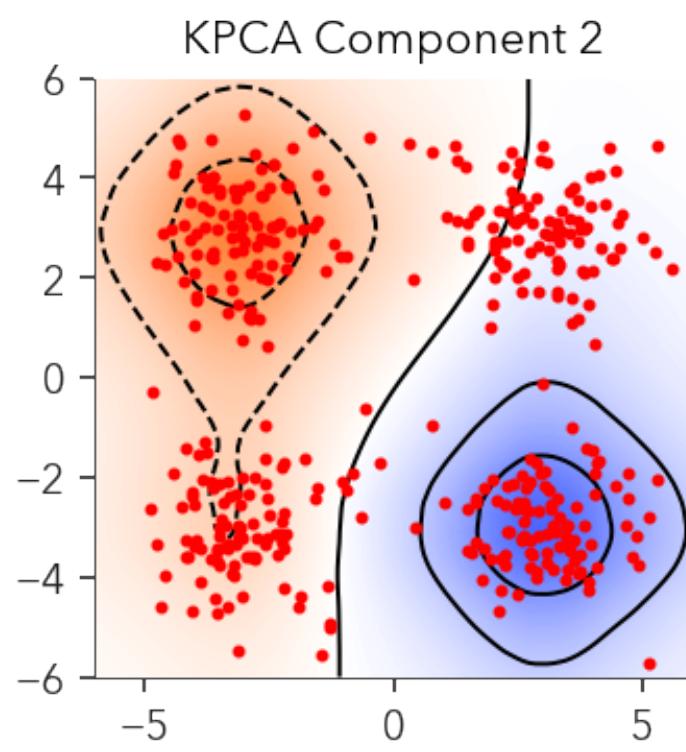
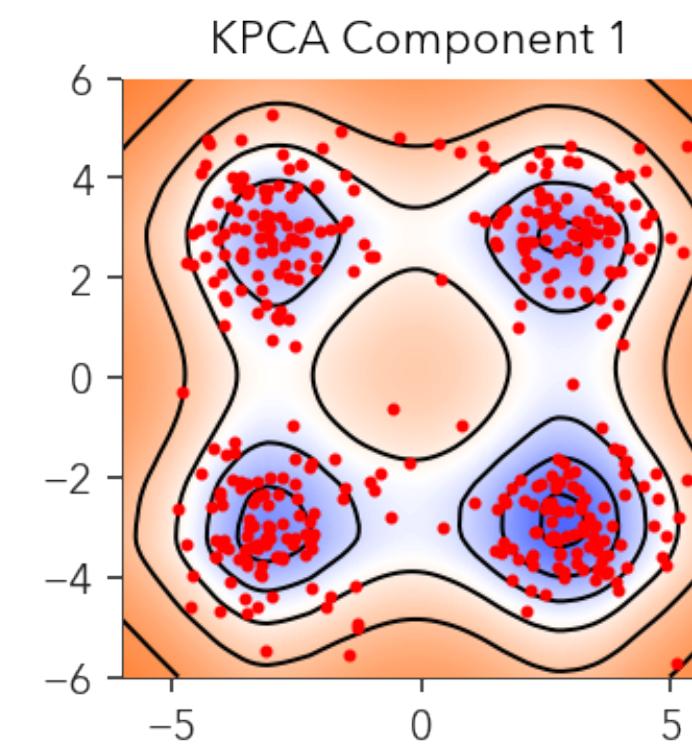


Contour shade shows product of each eigenvector with kernelized (x,y) coordinate

Varying results depending on the kernel

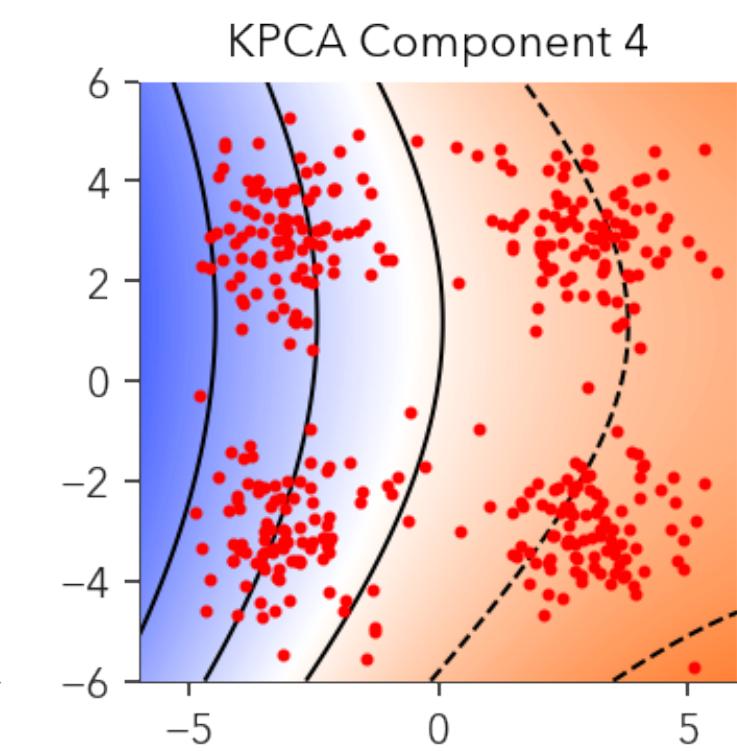
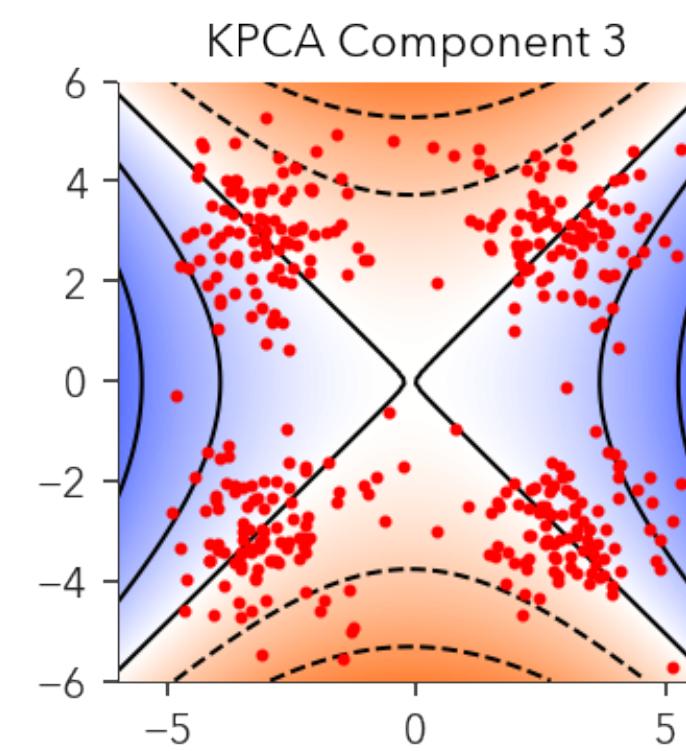
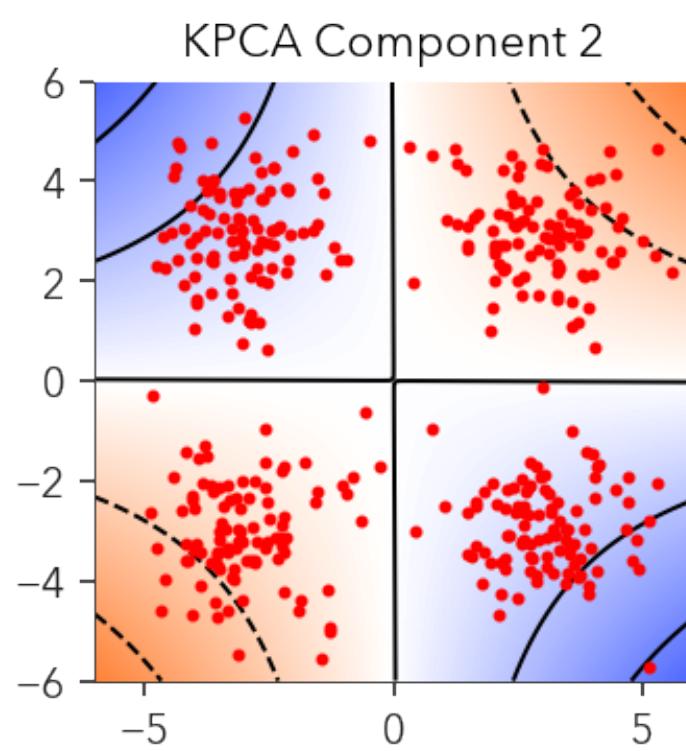
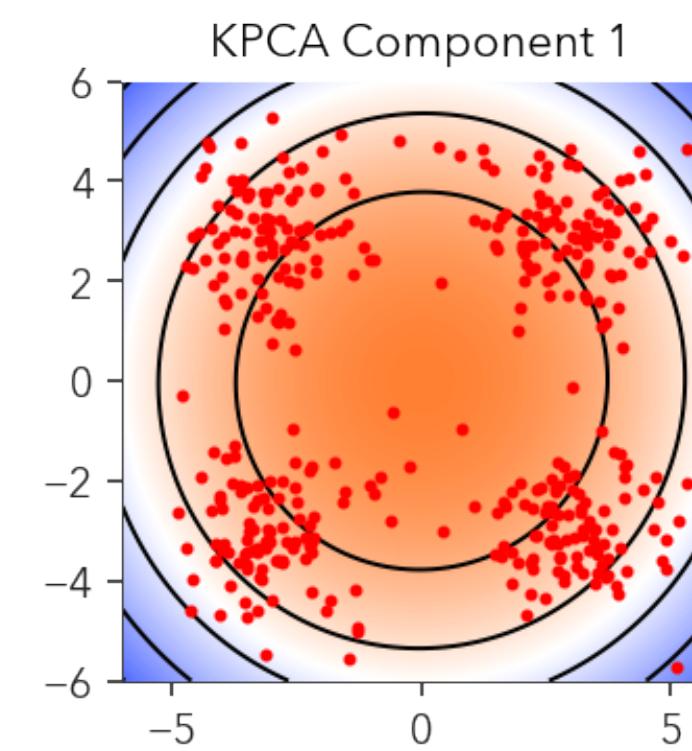
Laplacian kernel

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



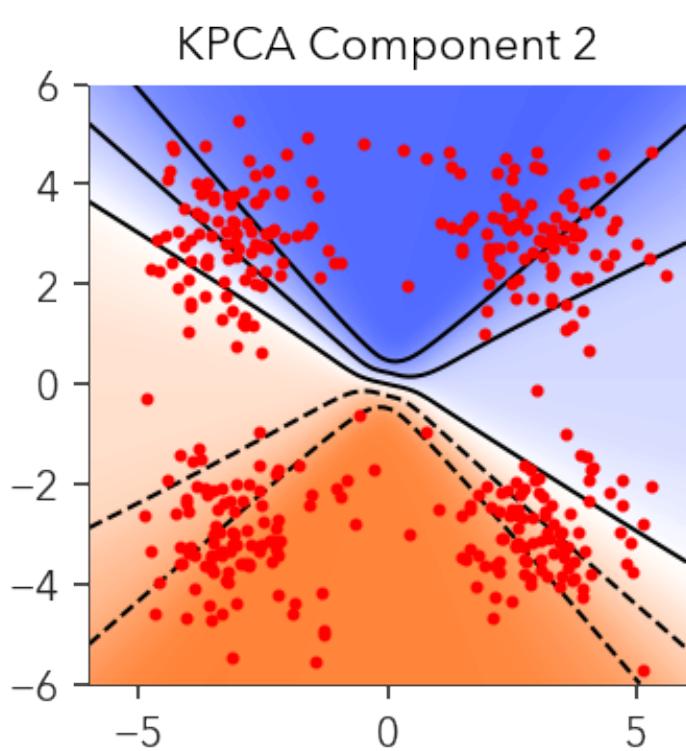
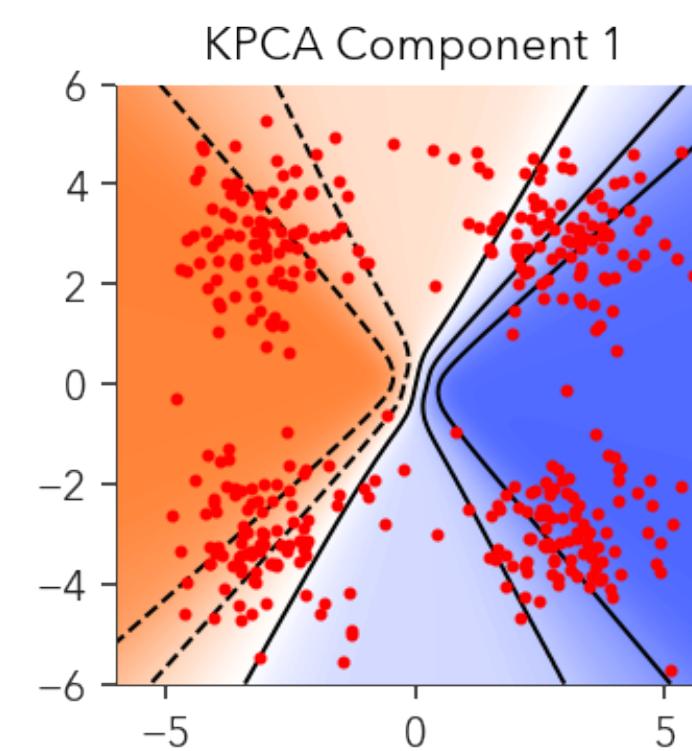
Polynomial kernel

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



Hyperbolic tangent kernel

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



Some KPCA notes

- The eigenanalysis is now bigger
 - \mathbf{K} is $N \times N$
 - Instead of $\mathbf{X} \cdot \mathbf{X}^T$ we analyze $\phi(\mathbf{X}^T \cdot \mathbf{X})$
 - 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

Distance matrix

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

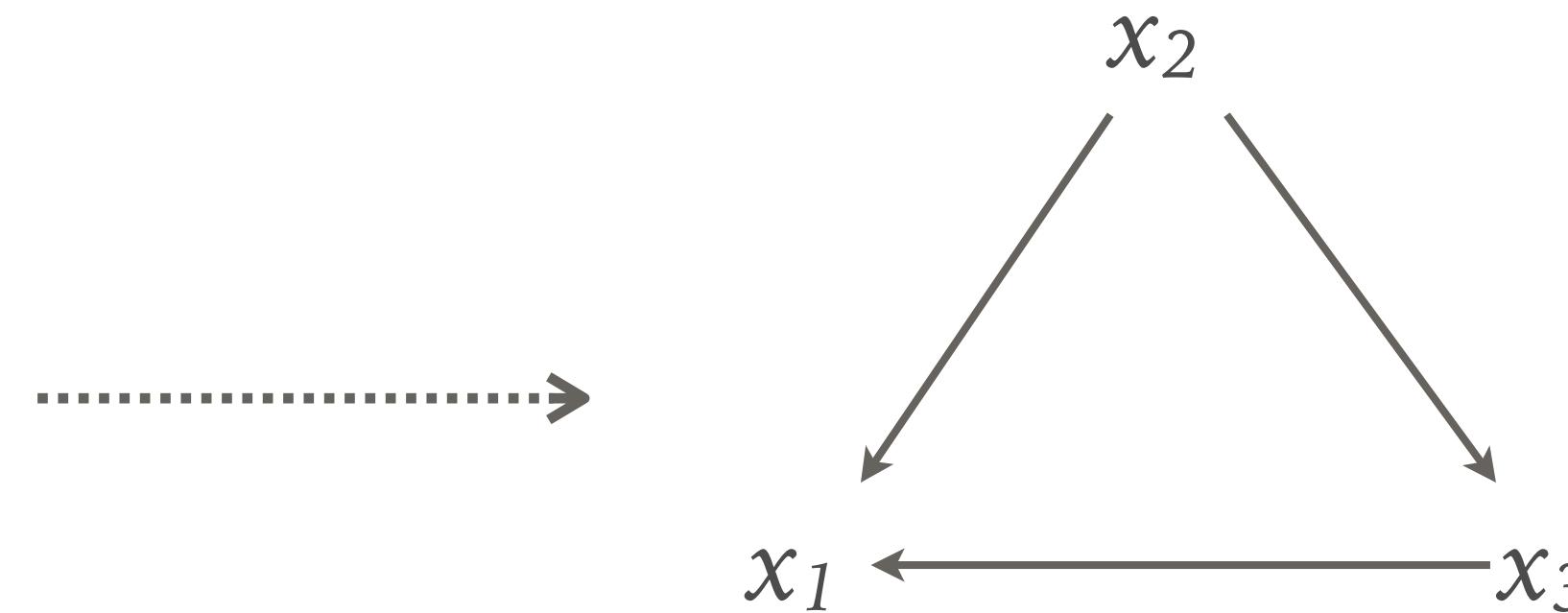
Reconstructing from distances

- 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:

$$d_{i,j} = \left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2 = \mathbf{x}_i^\top \cdot \mathbf{x}_i + \mathbf{x}_j^\top \cdot \mathbf{x}_j - 2\mathbf{x}_i^\top \cdot \mathbf{x}_j \Rightarrow$$
$$\mathbf{D} = \text{diag}^{-1} \left(\mathbf{X}^\top \cdot \mathbf{X} \right) \cdot \mathbf{1}_N^\top + \mathbf{1}_N \cdot \text{diag}^{-1} \left(\mathbf{X}^\top \cdot \mathbf{X} \right)^\top - 2\mathbf{X}^\top \cdot \mathbf{X}$$

$\mathbf{x}_i^\top \cdot \mathbf{x}_i$ replicated across columns $\mathbf{x}_j^\top \cdot \mathbf{x}_j$ replicated across rows

- 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:

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

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

- Applying on \mathbf{D} :

$$\begin{aligned} \mathbf{P} \cdot \mathbf{D} \cdot \mathbf{P} &= \mathbf{P} \cdot \text{diag}^{-1}(\mathbf{X}^\top \cdot \mathbf{X}) \cdot \mathbf{1}_N^\top \cdot \mathbf{P} + \mathbf{P} \cdot \mathbf{1}_N \cdot \text{diag}^{-1}(\mathbf{X}^\top \cdot \mathbf{X}) \cdot \mathbf{P} - 2\mathbf{P} \cdot \mathbf{X}^\top \cdot \mathbf{X} \cdot \mathbf{P} = \\ &= -2\mathbf{P} \cdot \mathbf{X}^\top \cdot \mathbf{X} \cdot \mathbf{P} = -2(\mathbf{X} \cdot \mathbf{P})^\top \cdot (\mathbf{X} \cdot \mathbf{P}) = -2(\mathbf{X} - \bar{\mathbf{x}})^\top \cdot (\mathbf{X} - \bar{\mathbf{x}}) \end{aligned}$$

- Bonus, a zero mean assumption on \mathbf{x} (never a bad thing!)

Multidimensional Scaling (MDS)

- Get distances and “double center”:

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

- Estimate \mathbf{X} from S

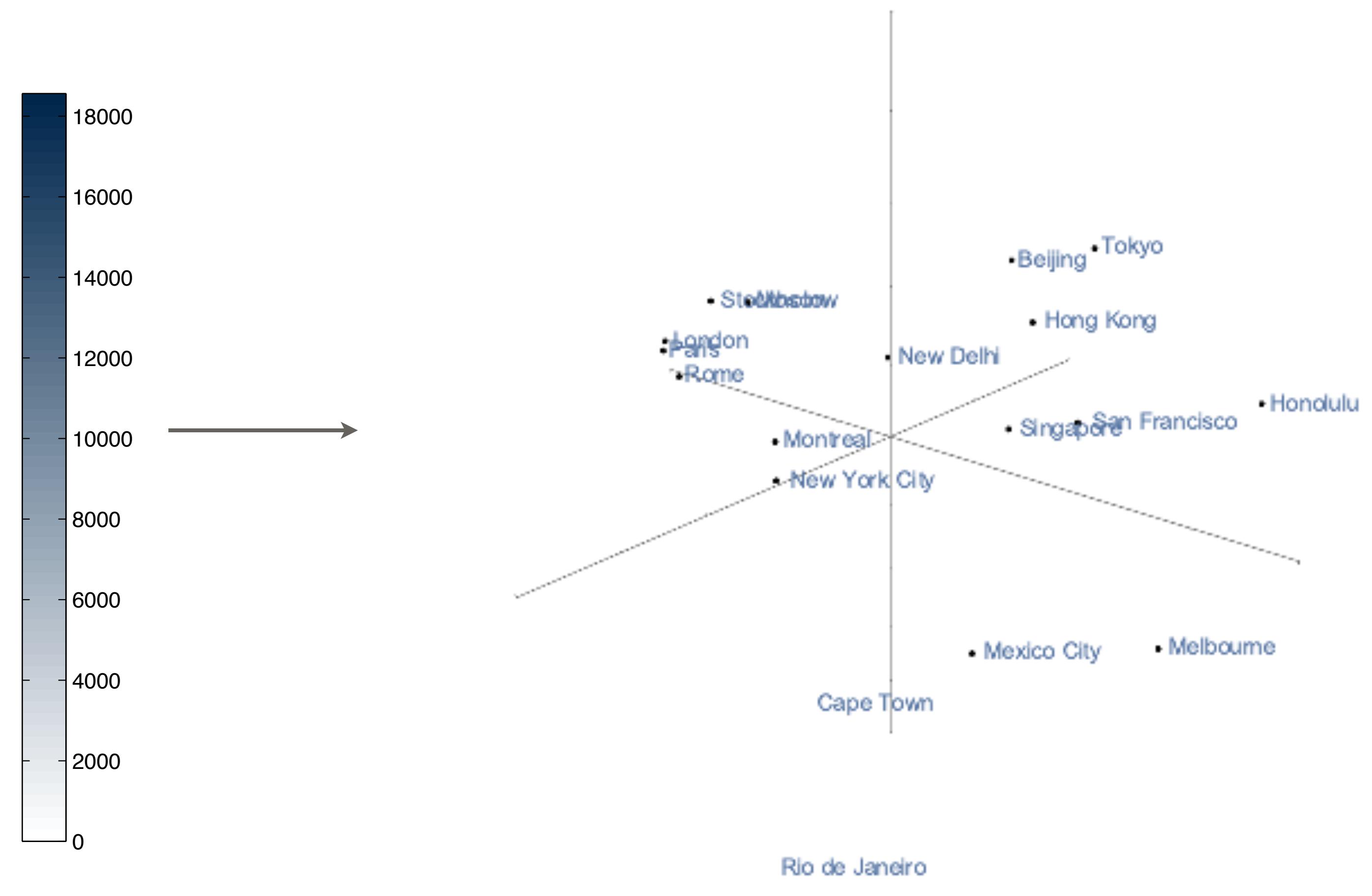
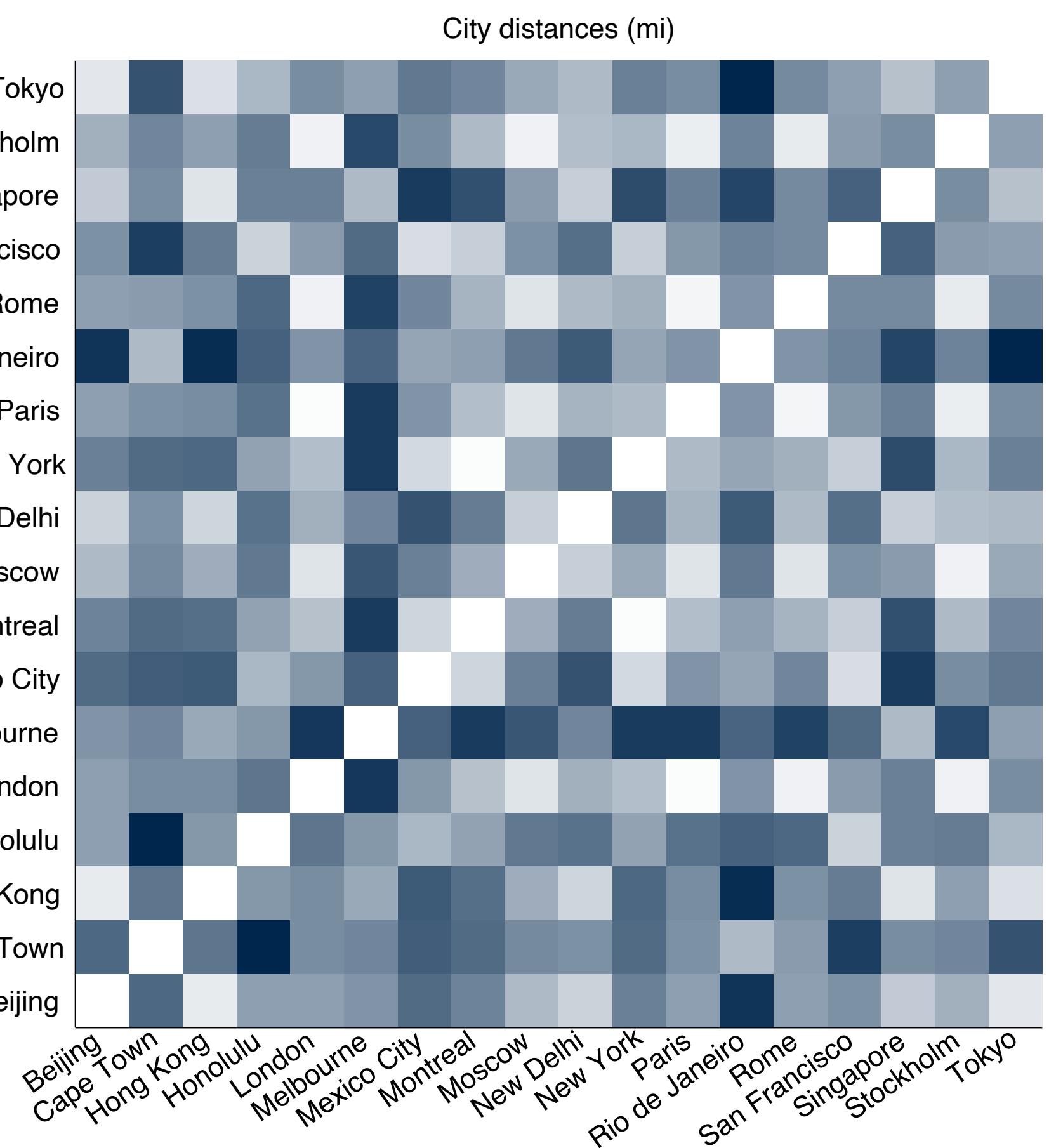
$$S = \mathbf{X}^\top \cdot \mathbf{X} \Rightarrow S = \mathbf{U} \cdot \Lambda \cdot \mathbf{U}^\top \Rightarrow \hat{\mathbf{X}} = \Lambda^{1/2} \cdot \mathbf{U}^\top$$

Eigendecomposition

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

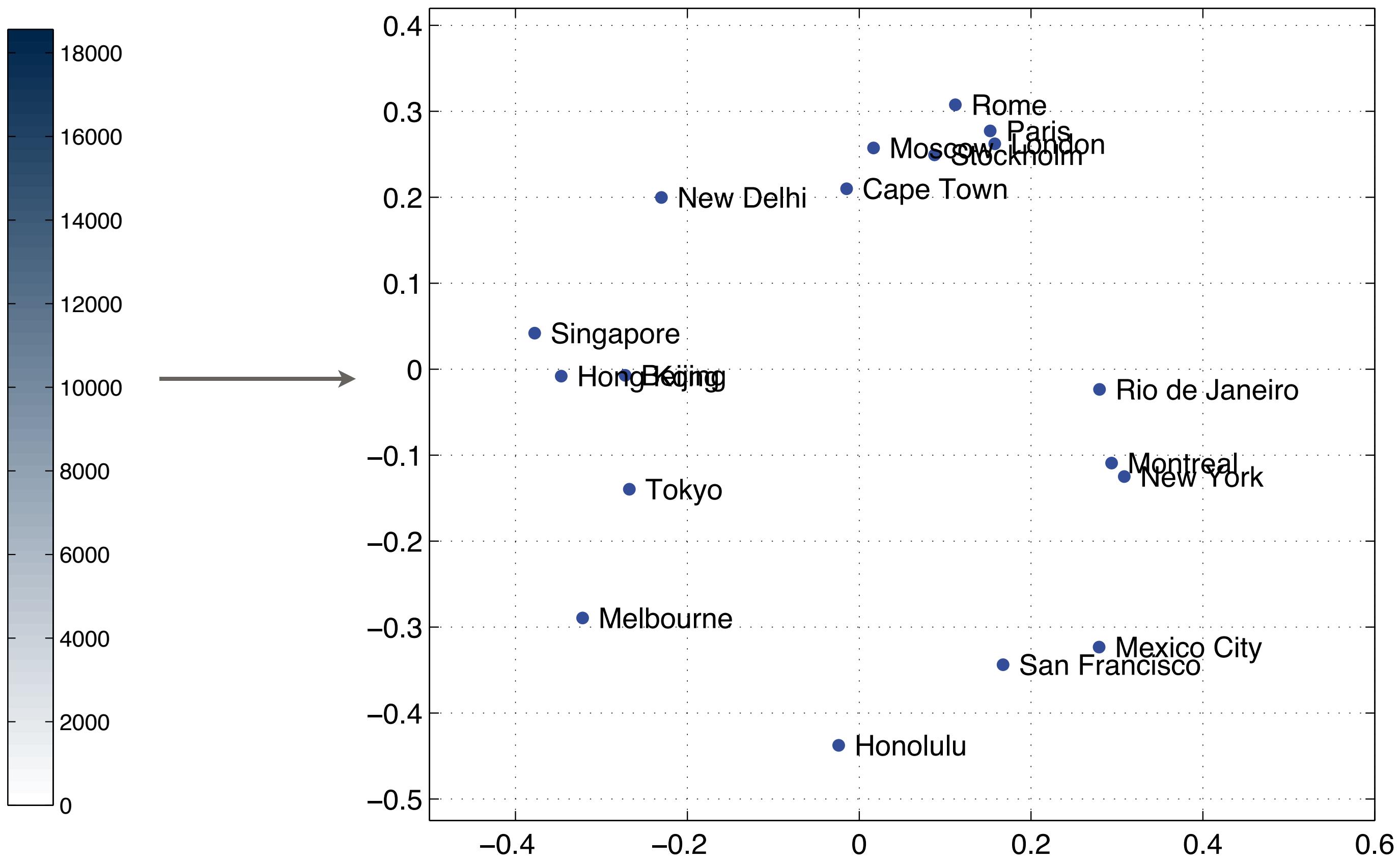
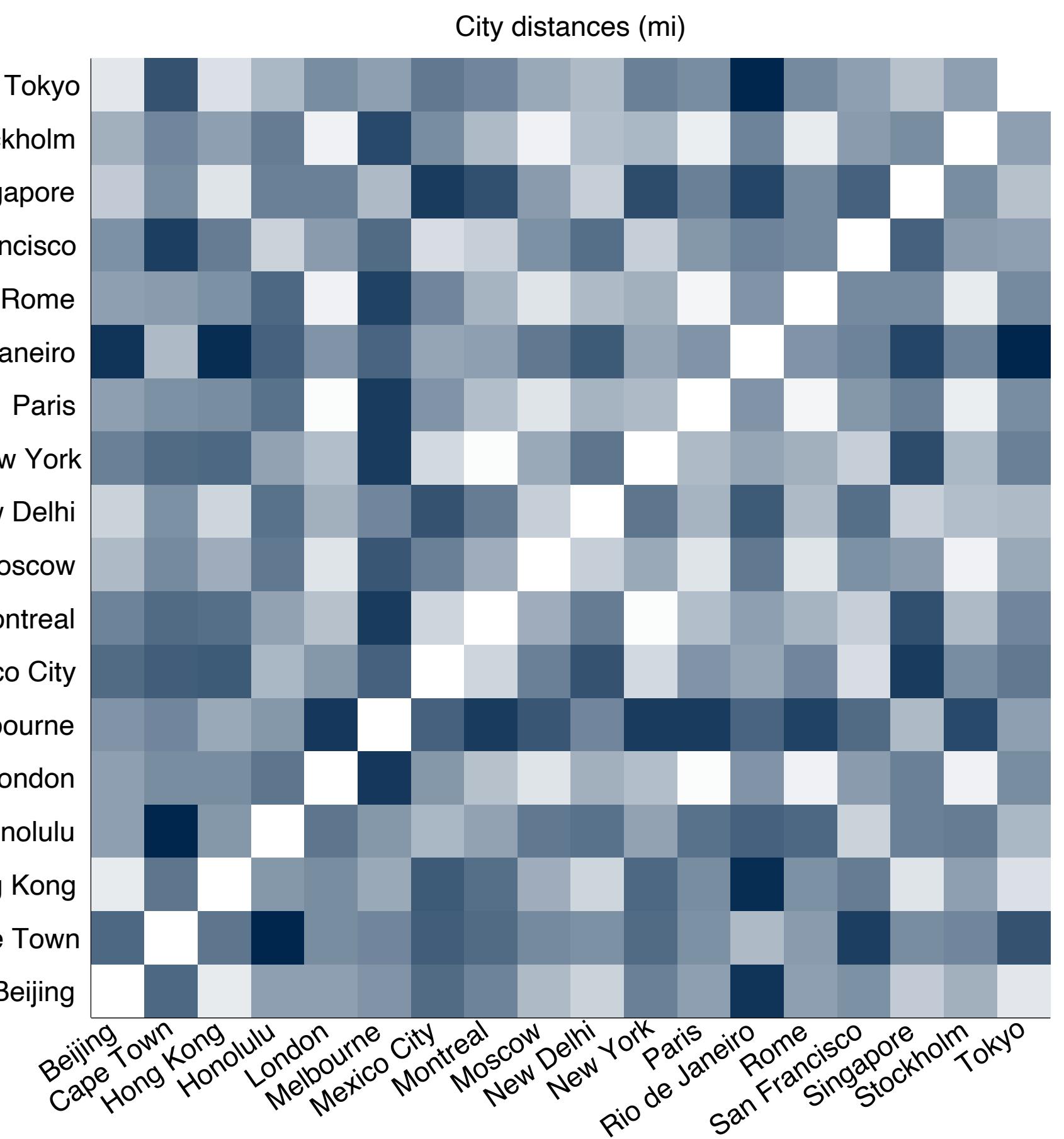
Back to the problem at hand

- City distance matrix results in an accurate map!



With fewer dimensions

- Also pretty accurate!

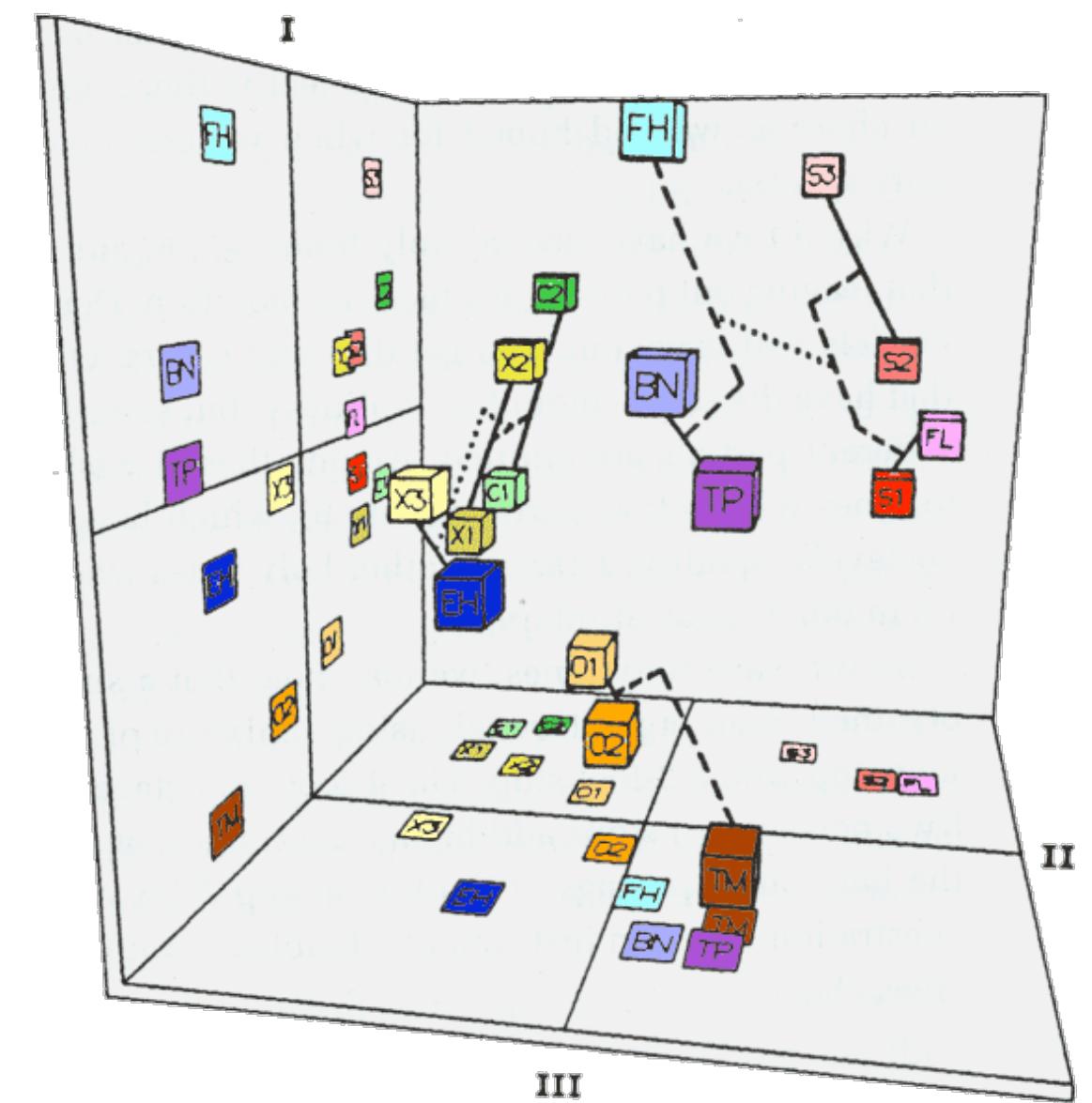


MDS variations

- 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!
 - Very useful for psychology experiments
- 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



Gray's timbre space

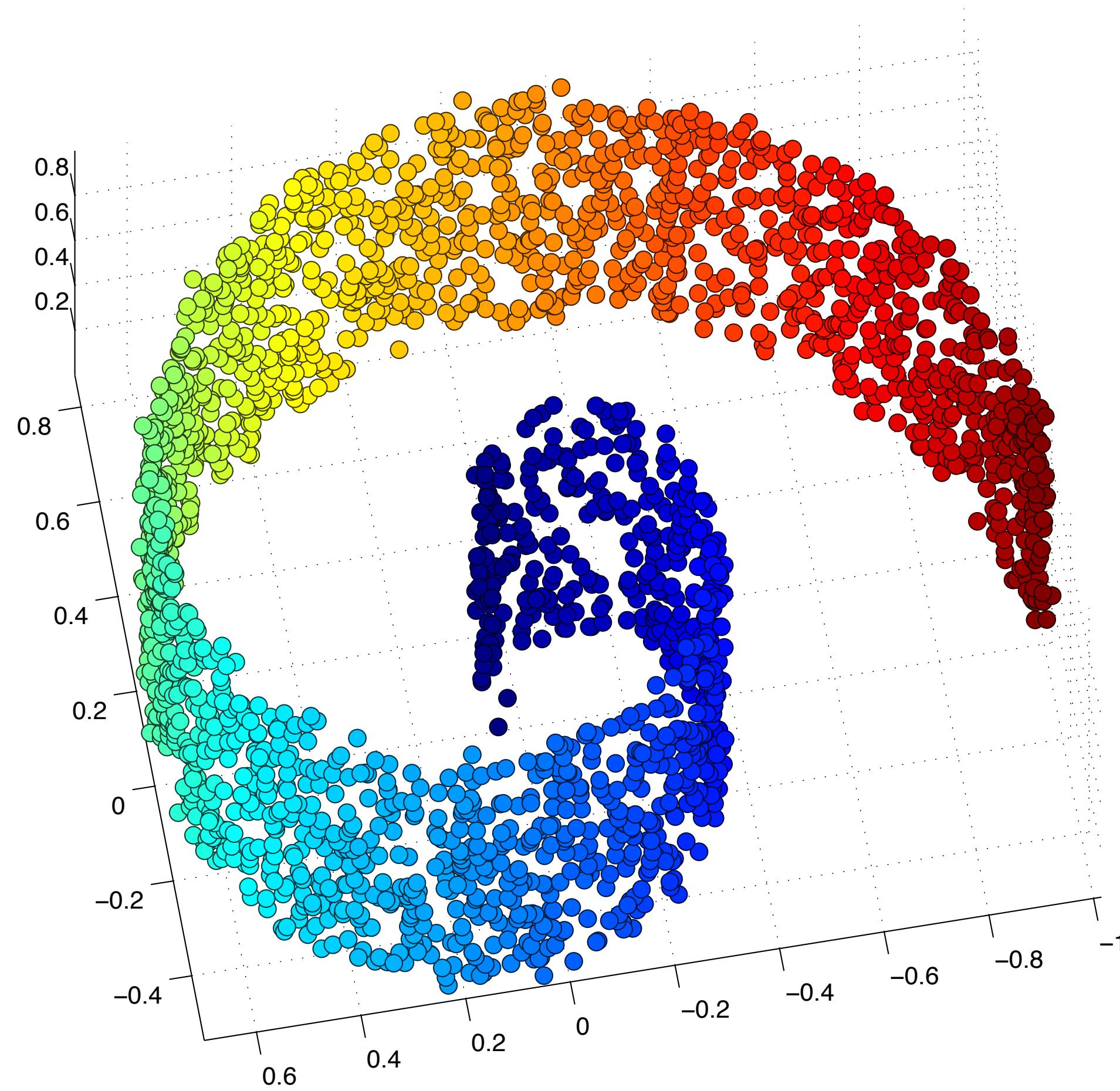
Data with low intrinsic dimension

- Actual dimension can be misleading

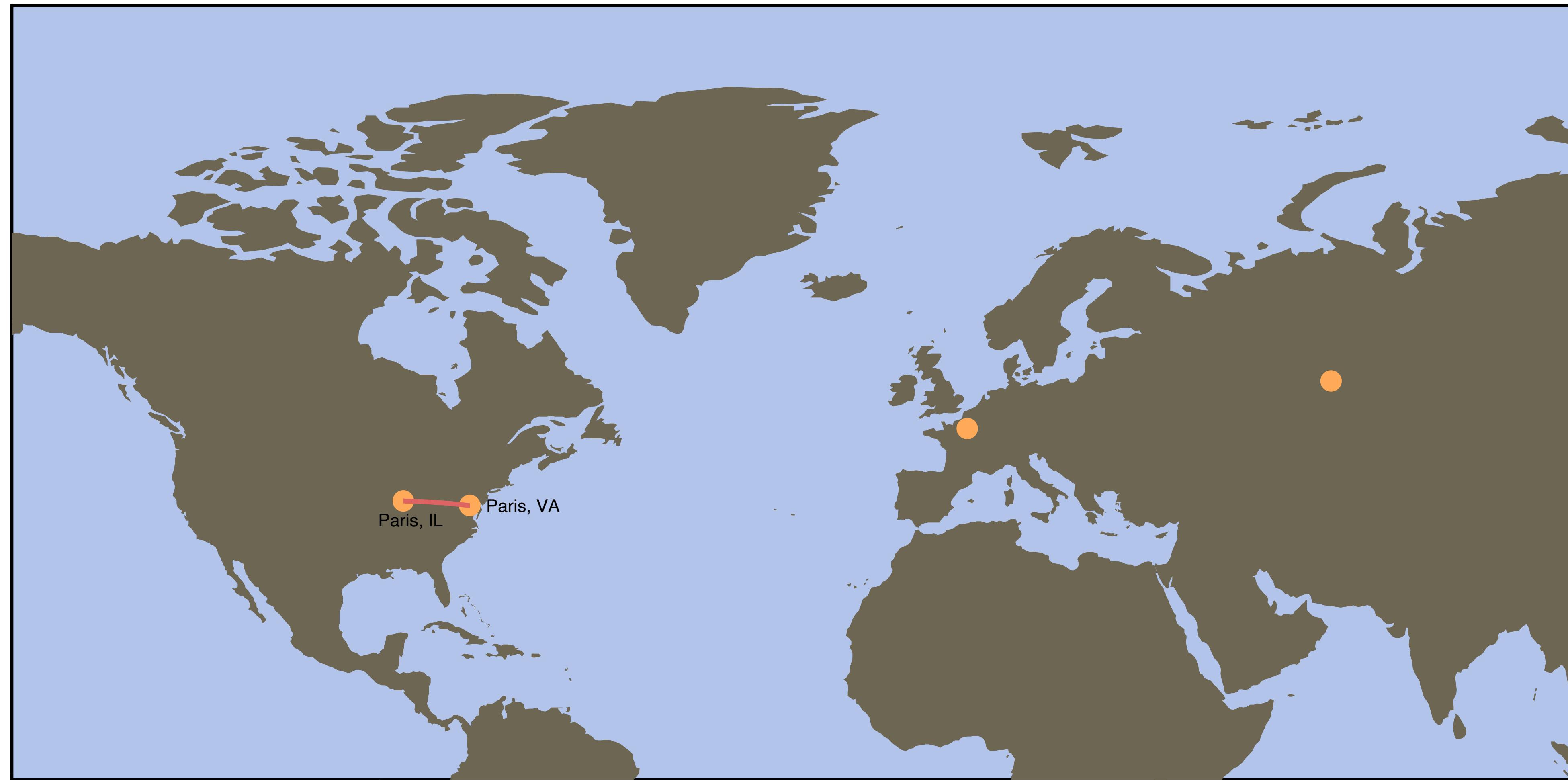


Data with low intrinsic dimension

- Actual dimension can be misleading



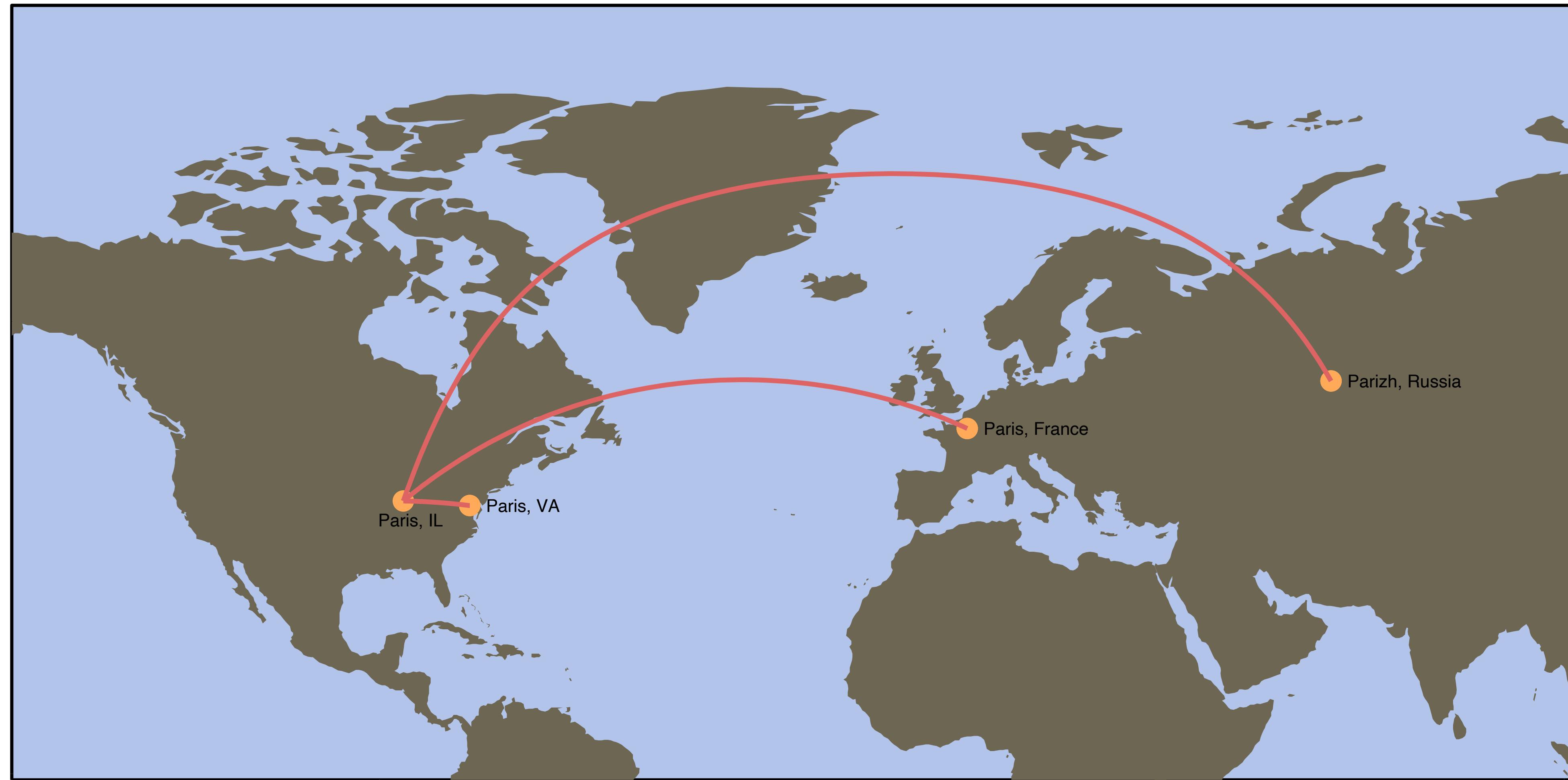
Local vs. Global



Local vs. Global



Local vs. Global



Preserving topology

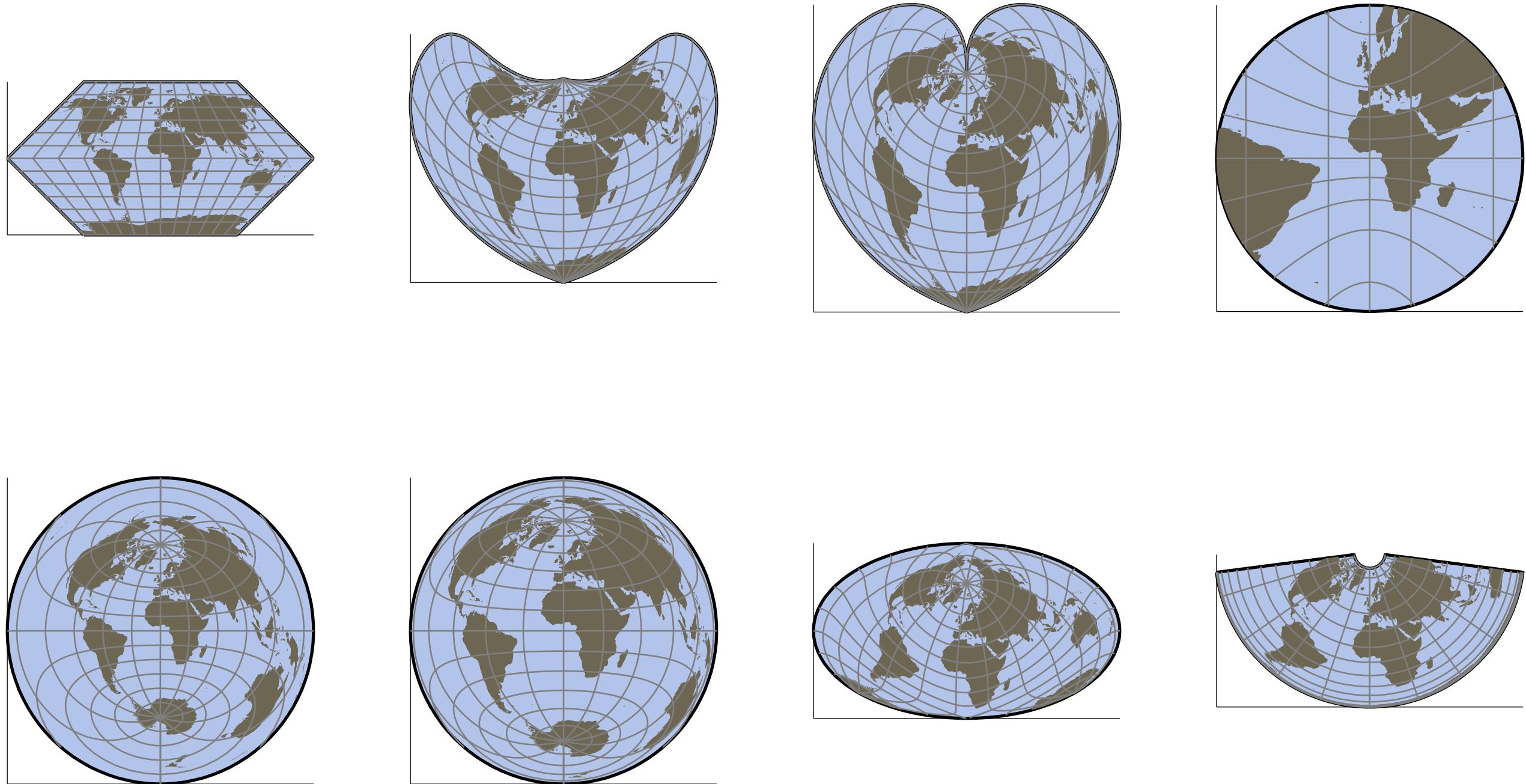
- Looking at the neighboring structure
 - Earth is round, but locally flat
- Reduce dimensionality while keeping the neighborhood structure unchanged

Example: Map projections

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

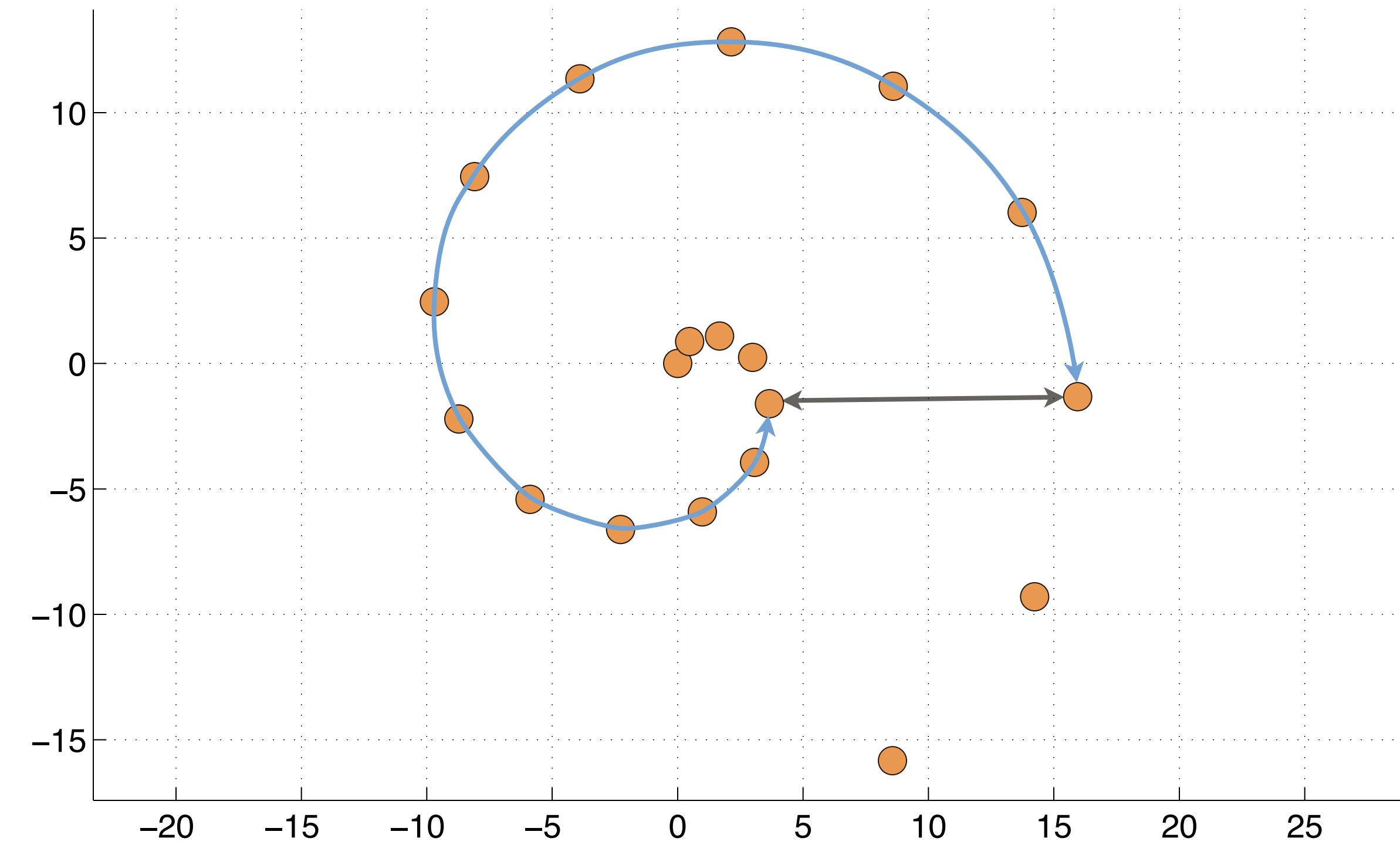


*Dimensionality
reduction*



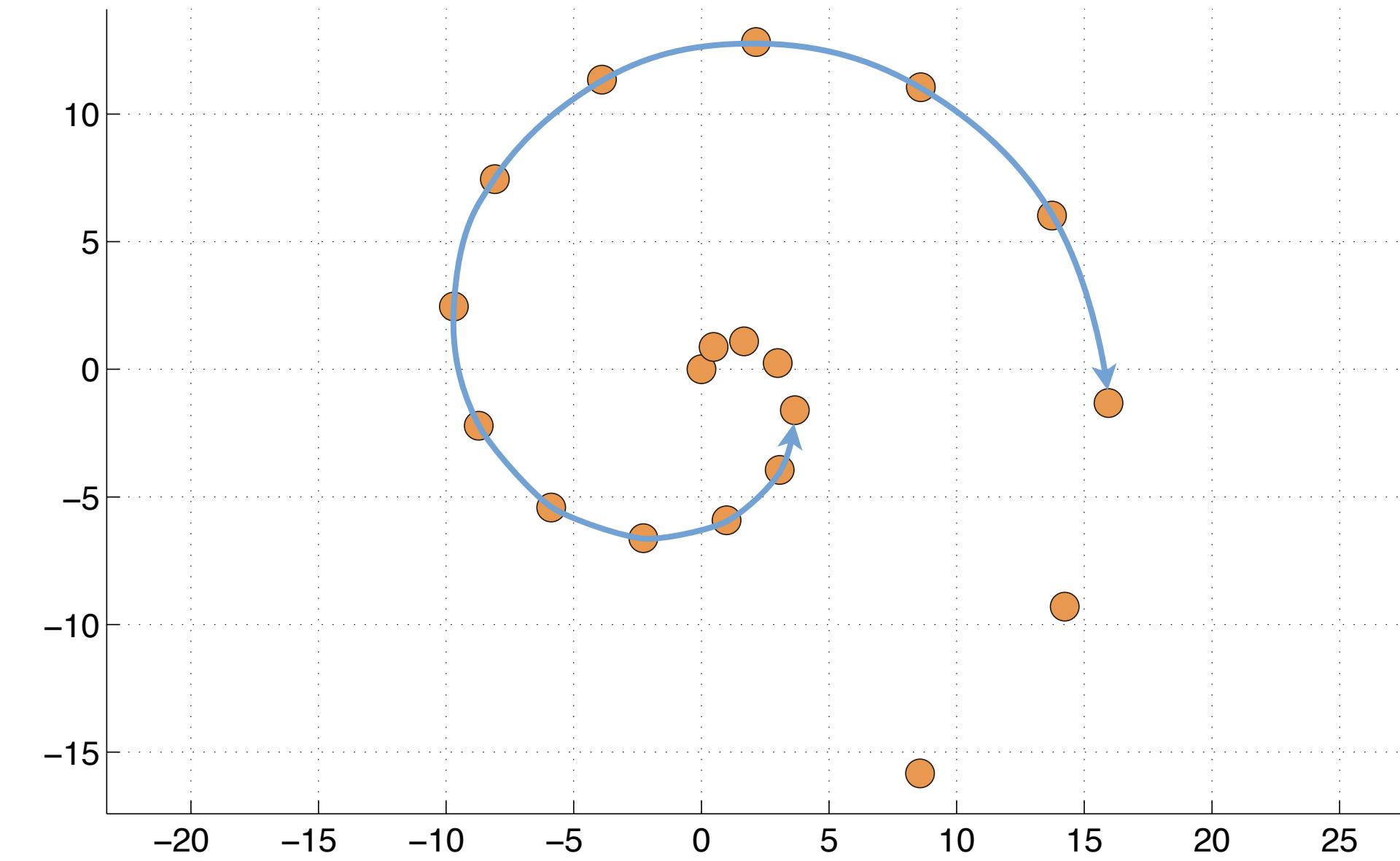
Euclidean vs. Geodesic distance

- On a manifold, distance can mean many things
 - Euclidean distance ignores manifold structure
 - Geodesic distance is more appropriate

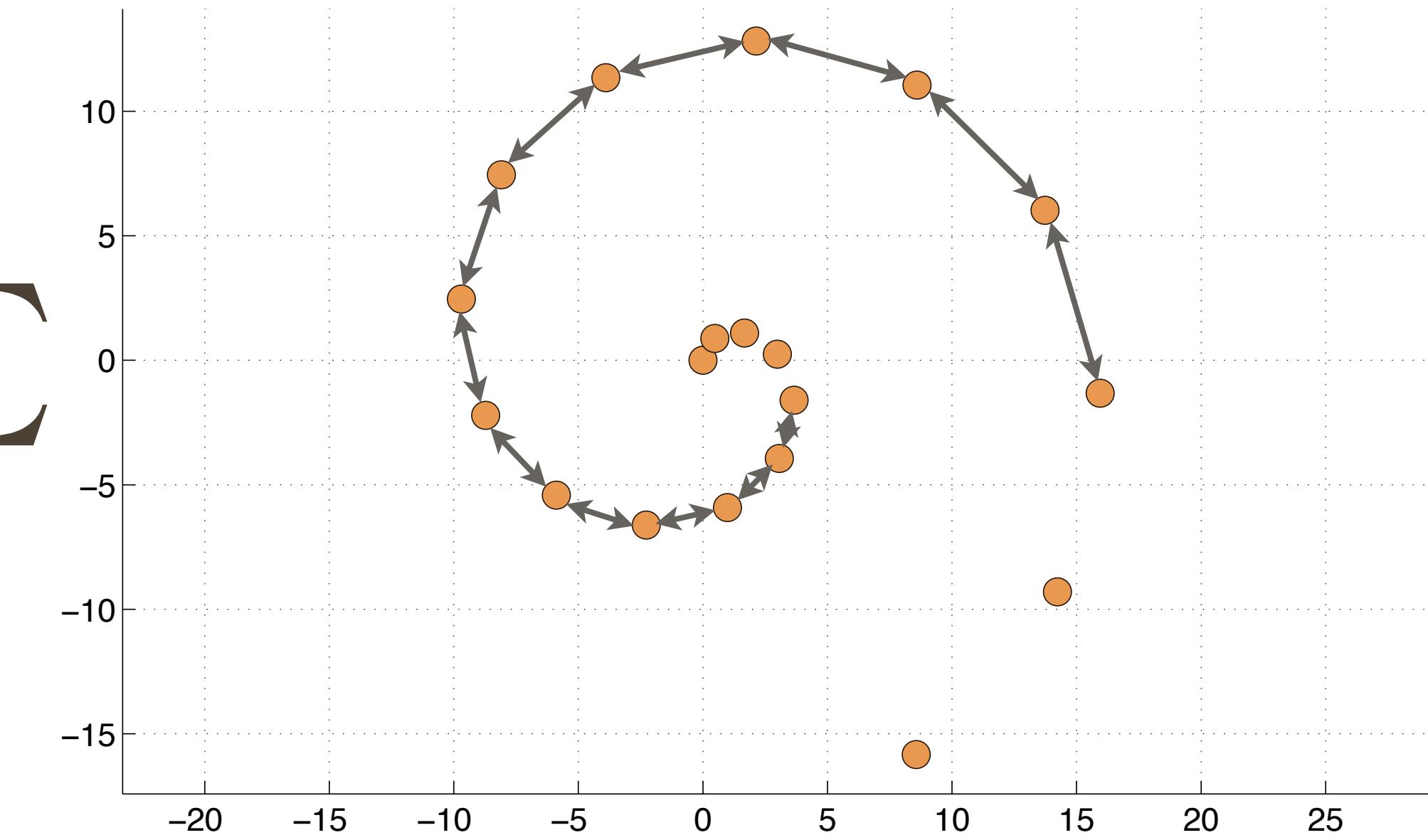


Getting the distance using just data

- 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



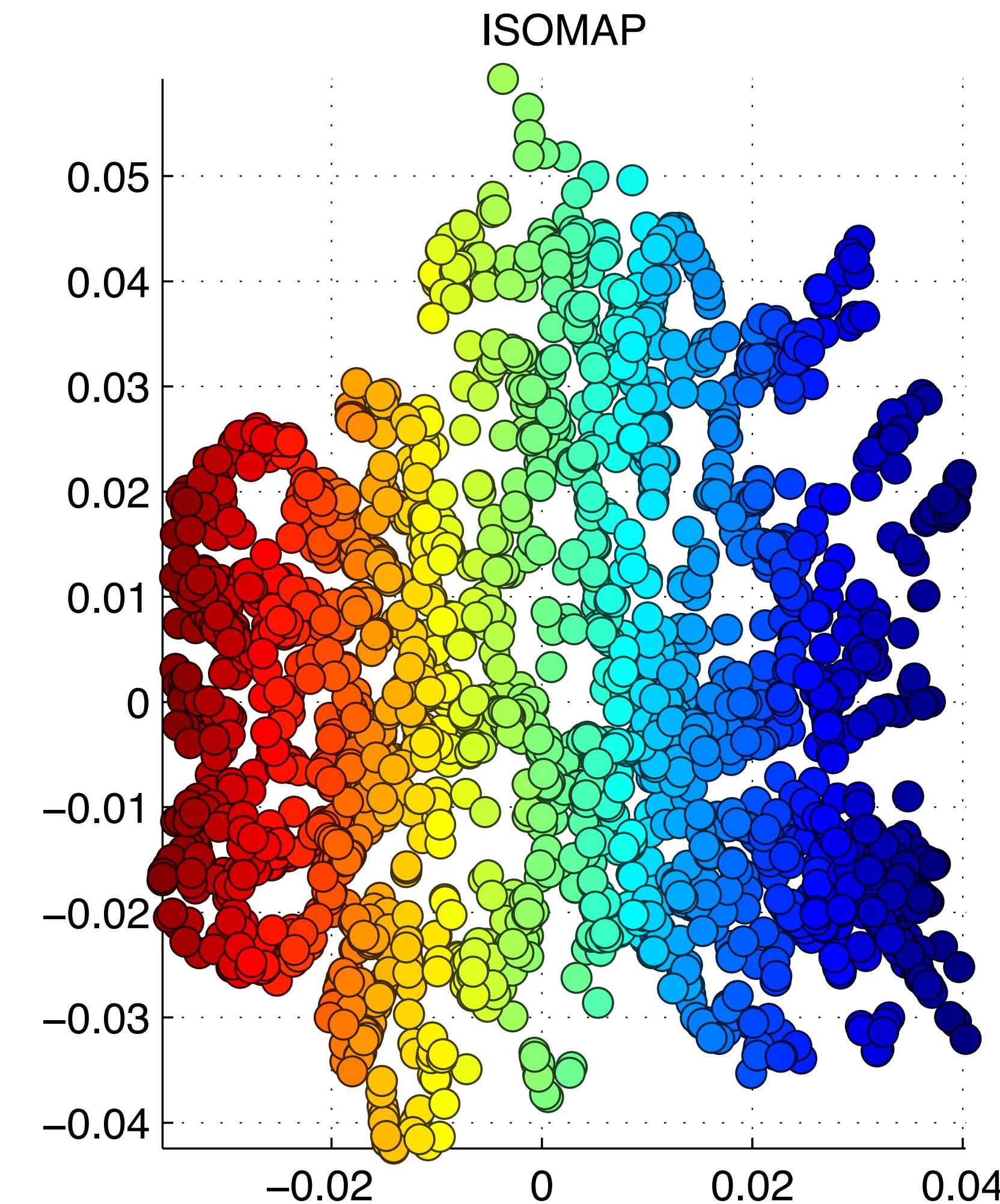
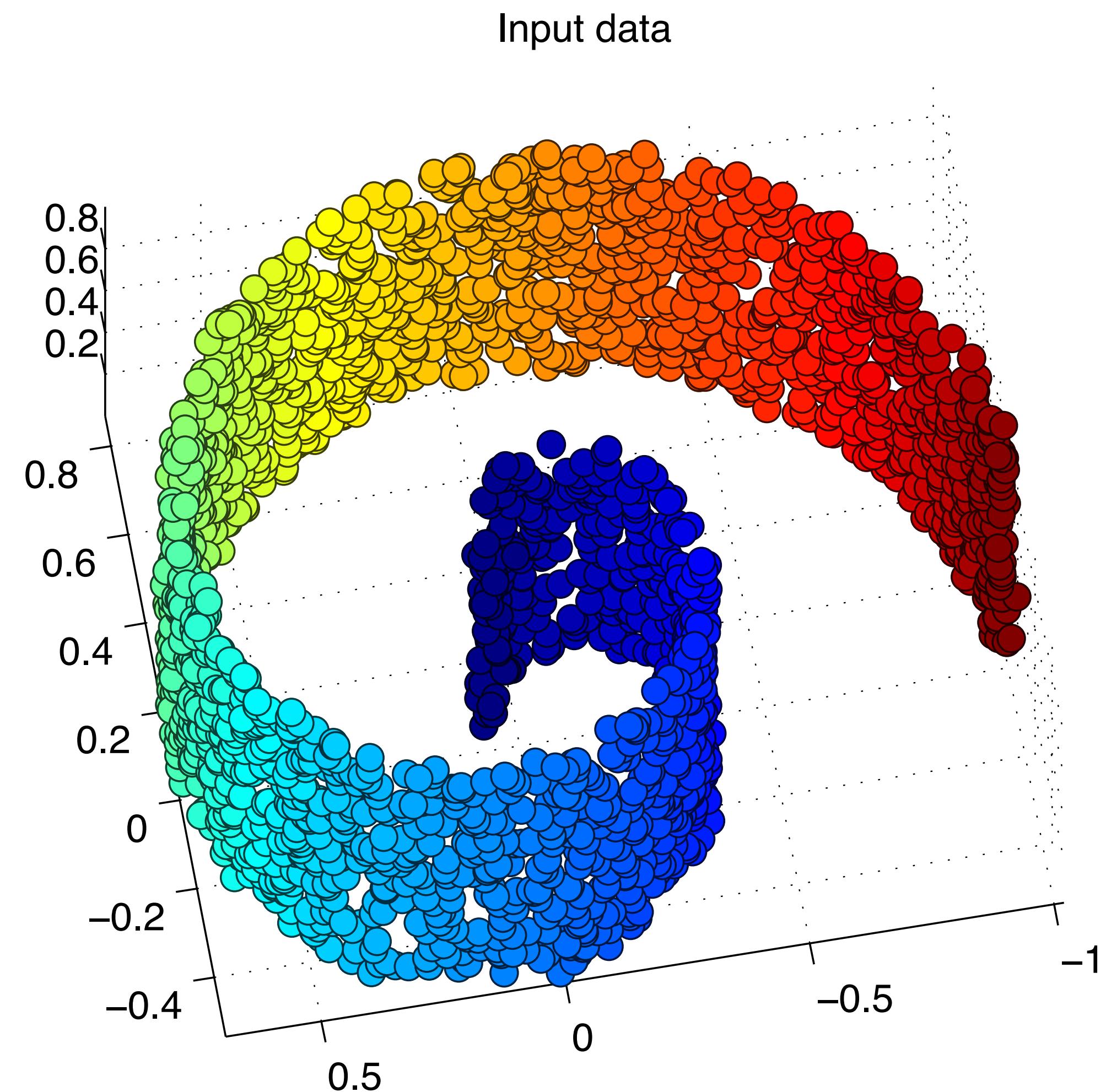
$$\| \cdot \|_2^2 \sum$$



ISOMAP

- Perform MDS using geodesic distances
 - Distance matrix \mathbf{D} now contains geodesic distances
- Results in an *embedding* of our data that exposes its local geometry
 - Projects to Euclidean geometry layout that is proportional to the geodesic distances between data

On the Swiss roll



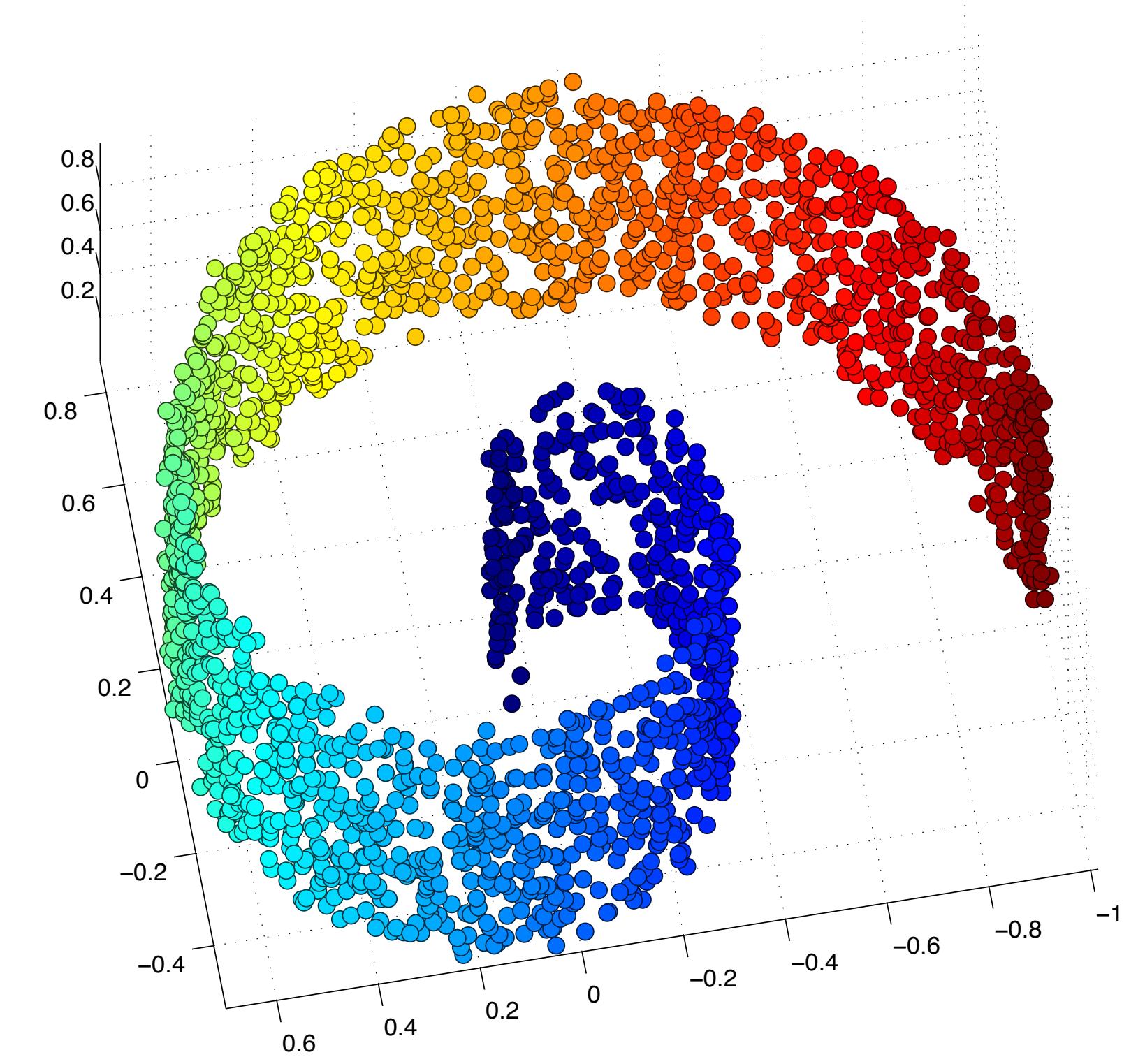
Another approach

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

$$\mathbf{x}_i \approx \sum_{j \in N(i)} w_{i,j} \mathbf{x}_j, \quad \sum_j w_{i,j} = 1$$

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

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

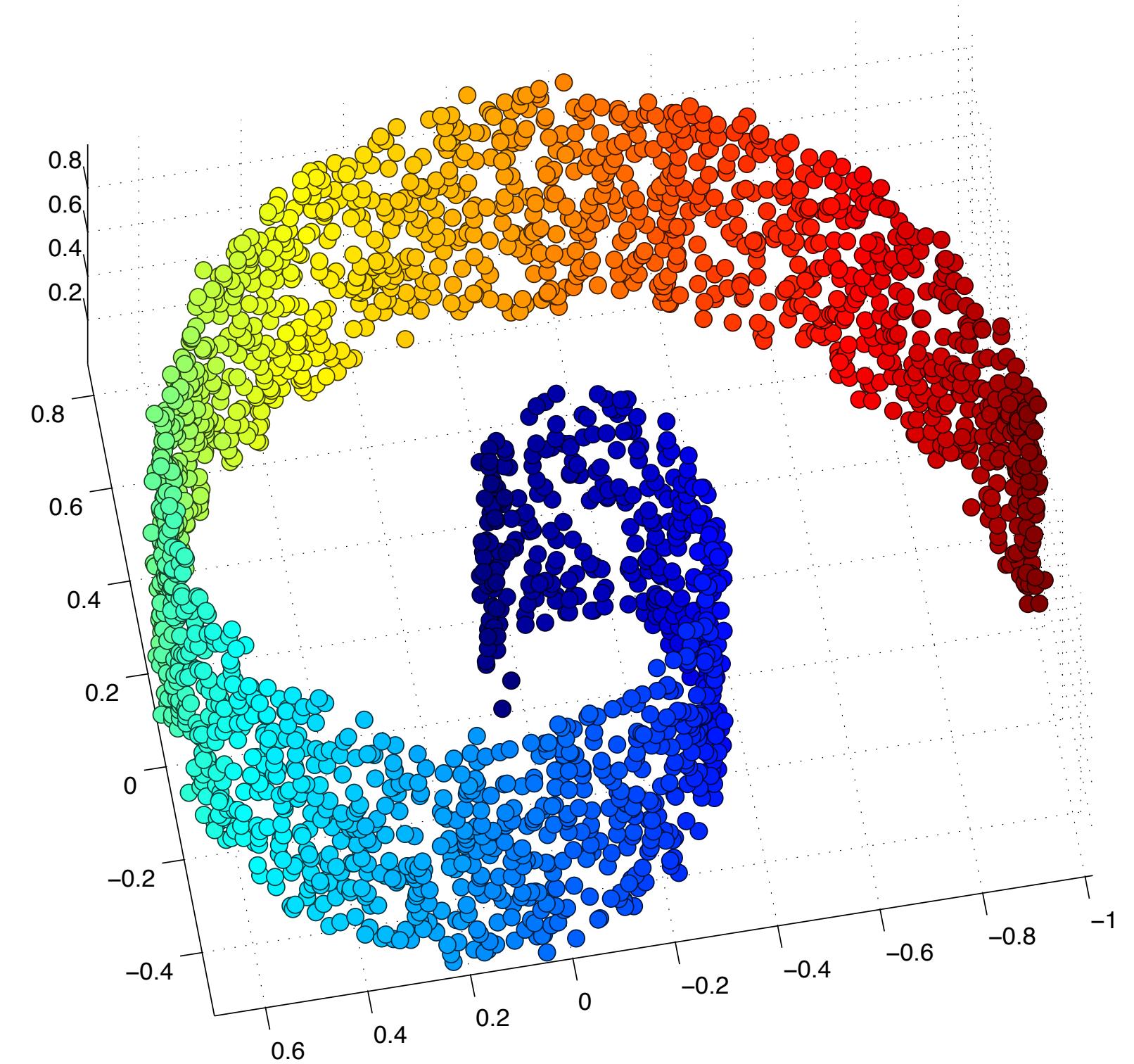


This should hold over other embeddings

- The weights will work just as well in a lower dimensional projection y
 - 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 y that minimizes the above expression



How to find y

- We can rewrite the cost function as: $\|Y - Y \cdot W\|$
 - W will have a zero diagonal, and be sparse
- If we expand it we get:

$$Y \cdot Y^\top - Y \cdot W^\top \cdot Y^\top - Y \cdot W \cdot Y^\top - Y \cdot W^\top \cdot W \cdot Y^\top$$

- We will impose: $Y \cdot Y^\top = I$
 - Since the scale of Y doesn't matter
- Which becomes: $Y \cdot (I - W)^\top \cdot (I - W) \cdot Y^\top$

Looks like a familiar form?

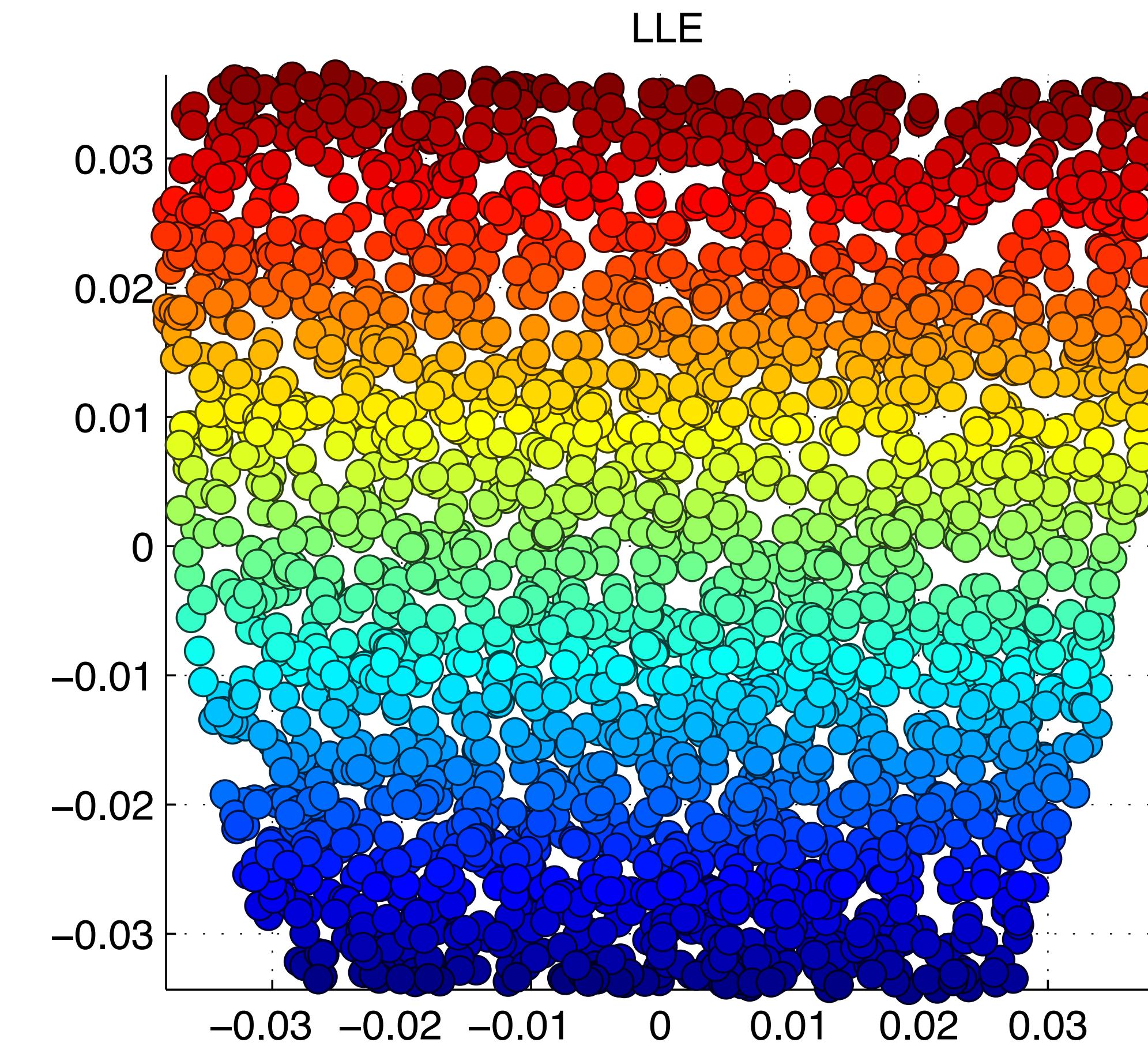
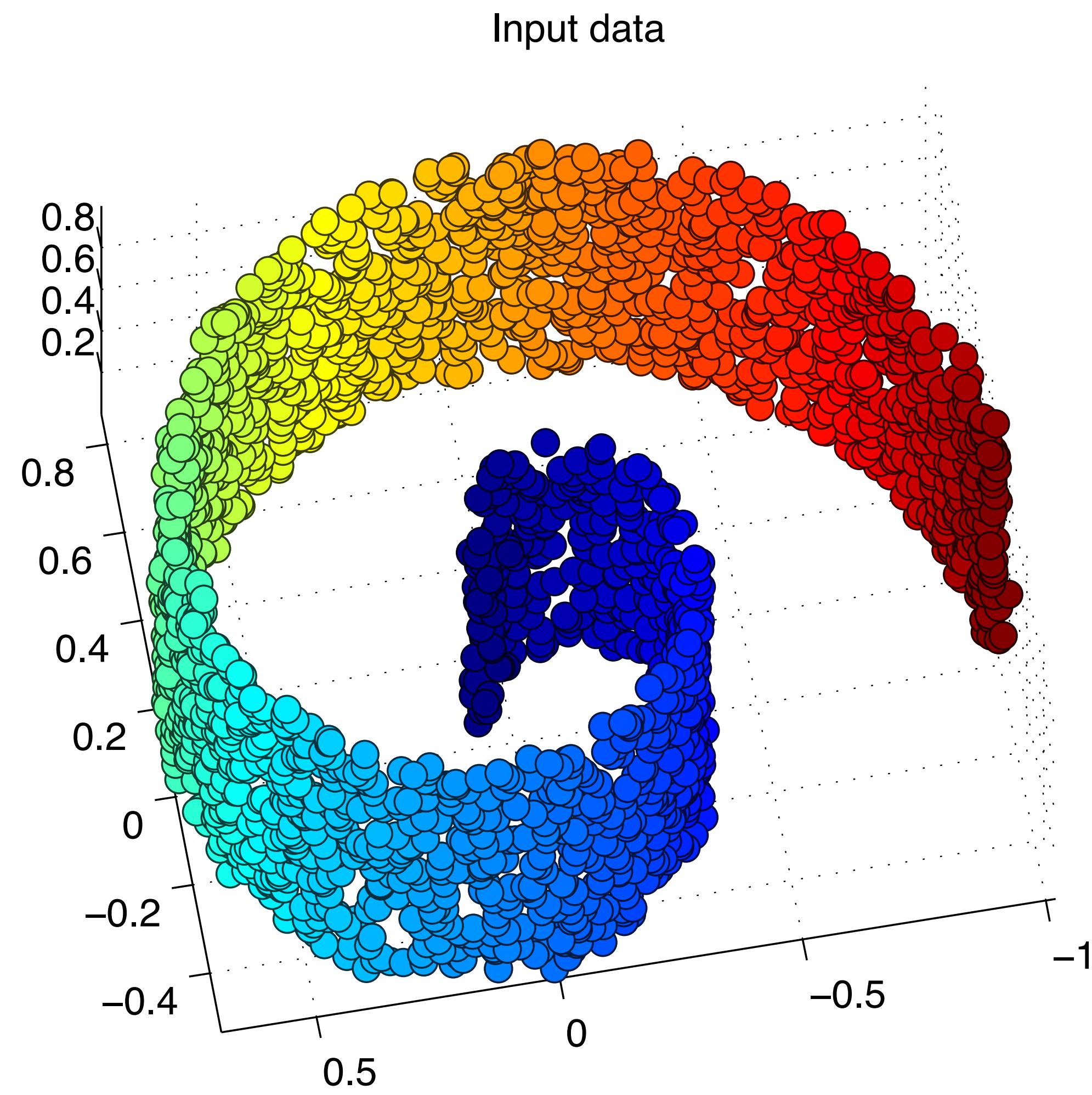
Let's get some eigenvectors again!

- We want a \mathbf{Y} that minimizes that product
- If we write it as an eigendecomposition

$$\mathbf{Y} \cdot (\mathbf{I} - \mathbf{W})^\top \cdot (\mathbf{I} - \mathbf{W}) \cdot \mathbf{Y}^\top = \mathbf{Y} \cdot \mathbf{M} \cdot \mathbf{Y}^\top = \Lambda$$

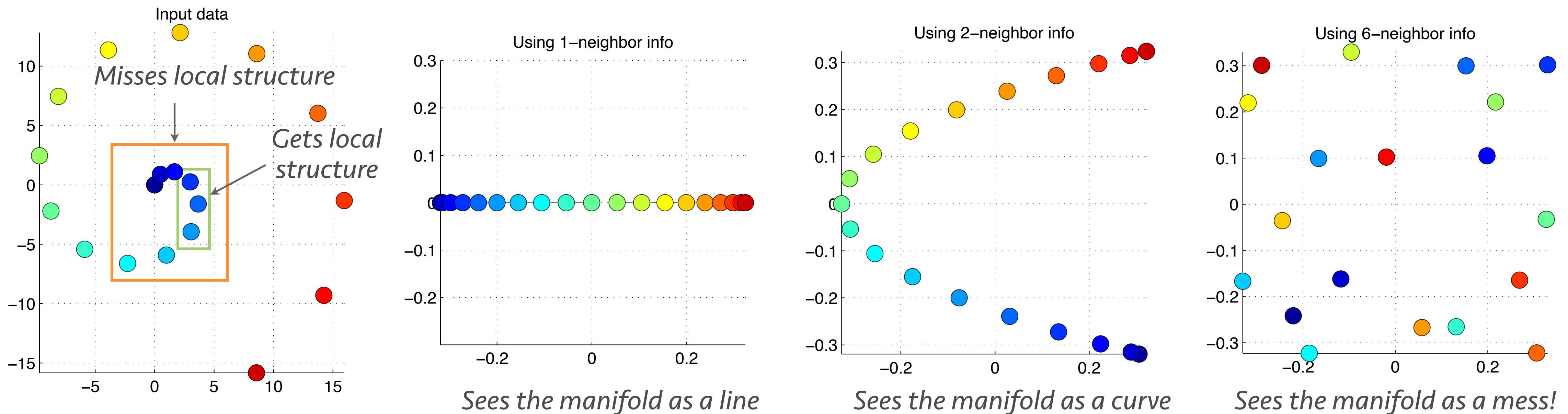
- And we know that the last eigenvalues will be closer to 0
- So if we pick the last eigenvectors of \mathbf{M} we get \mathbf{Y} !
 - The last eigenvalue will be 0, so for a K -dimensional embedding \mathbf{Y} we will keep the last $K+1$ eigenvectors except for the last one

Swiss roll example



A note on manifold methods

- You need to define what “local” means
 - E.g. in ISOMAP set distances of distant points to ∞ , in LLE you can constrain the size of the local neighborhood

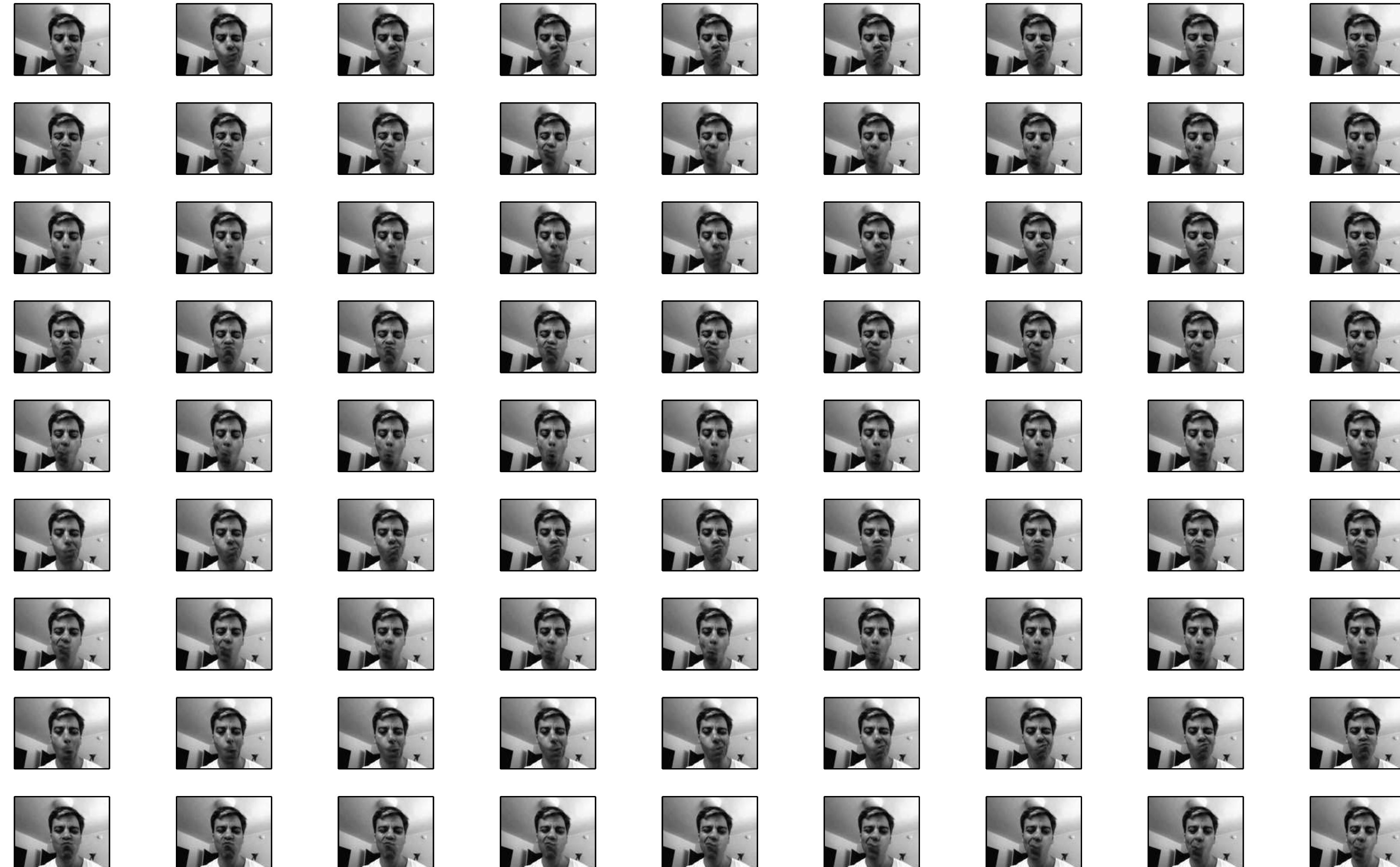


A video example

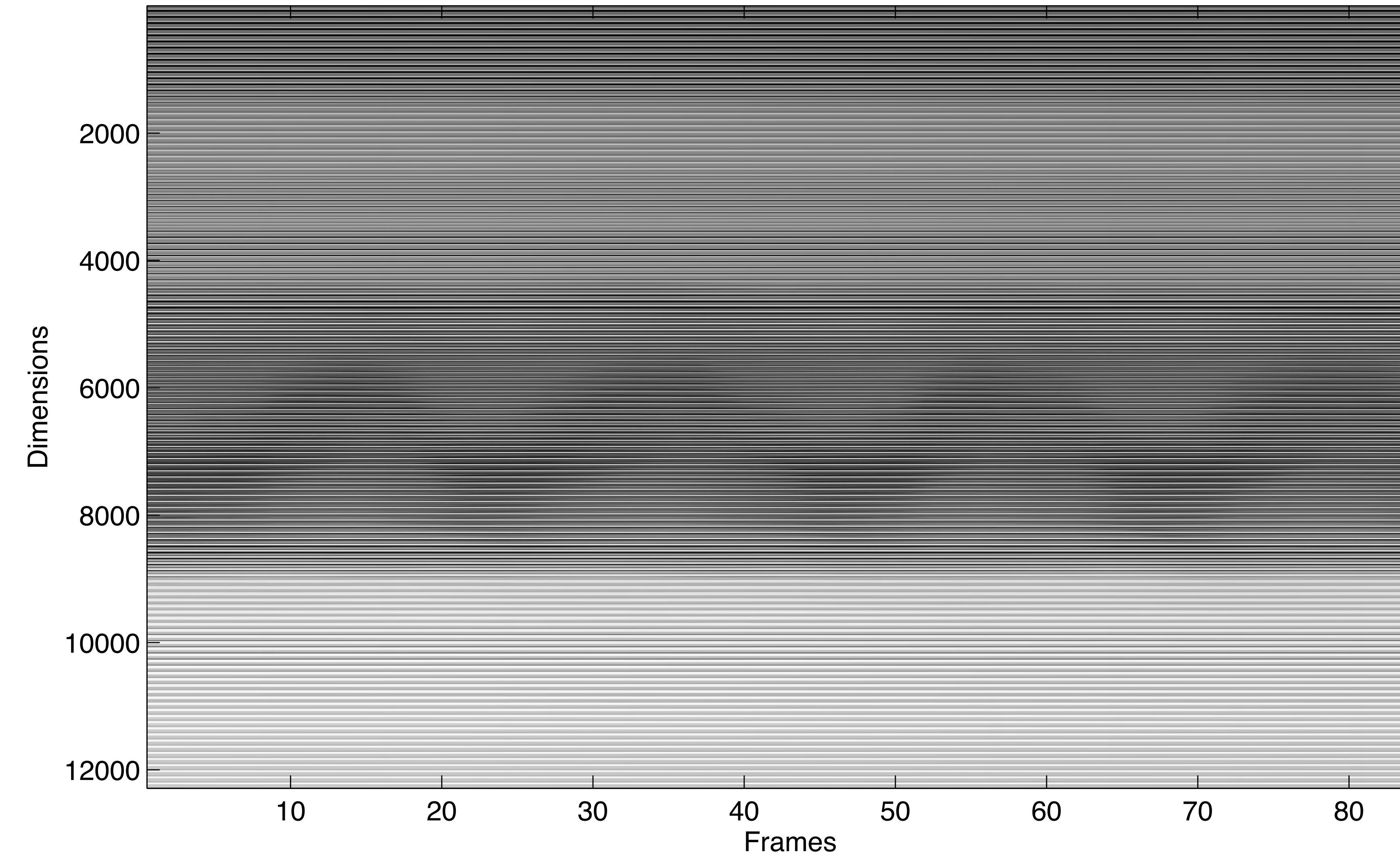
- A high dimensional input
 - 128×96 pixels = 12,288 dimensions
- Low dimensional structure
 - Moving lips around
- Can we simplify the data?



Each frame is a pose



When unraveled



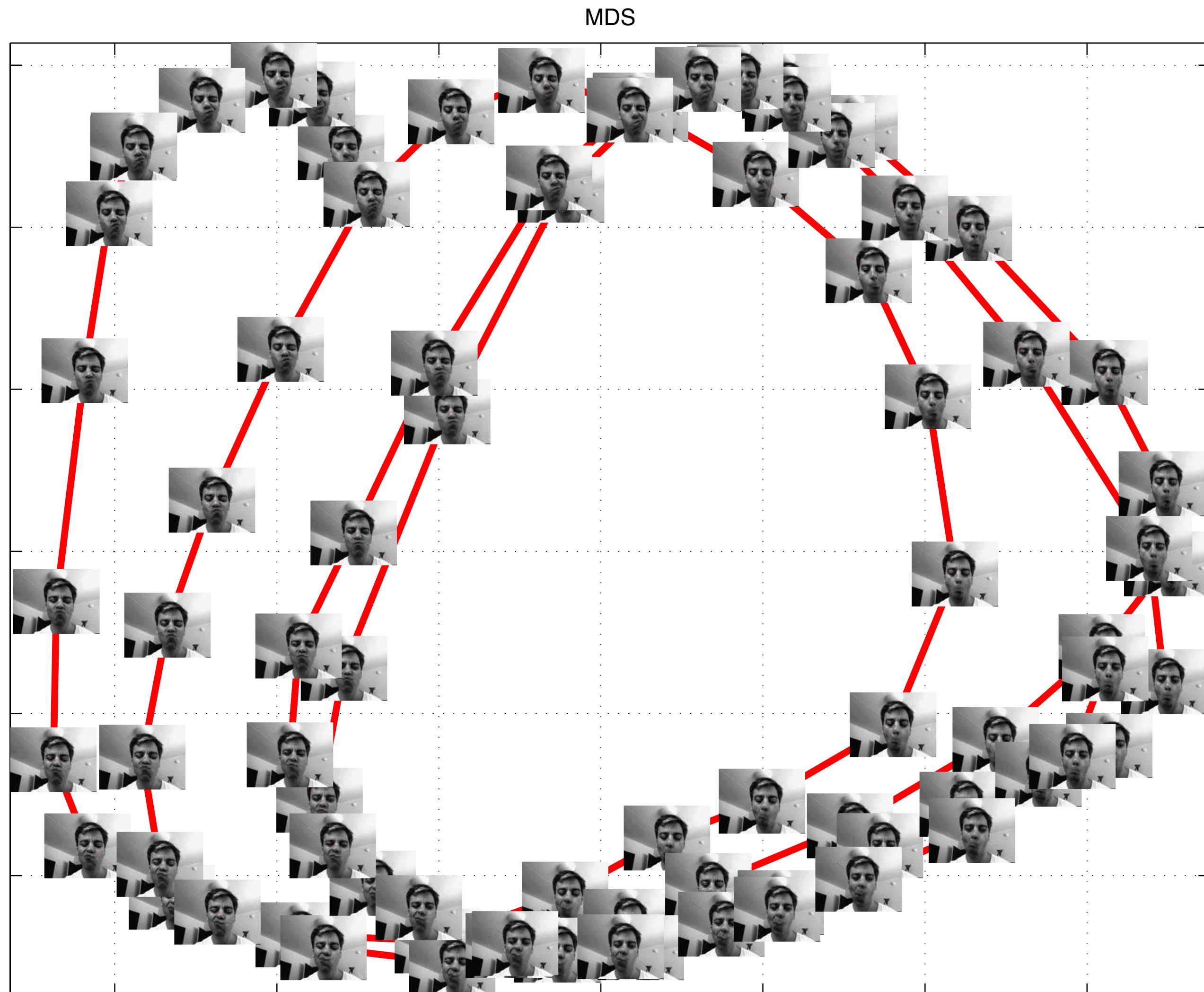
The manifold

- 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

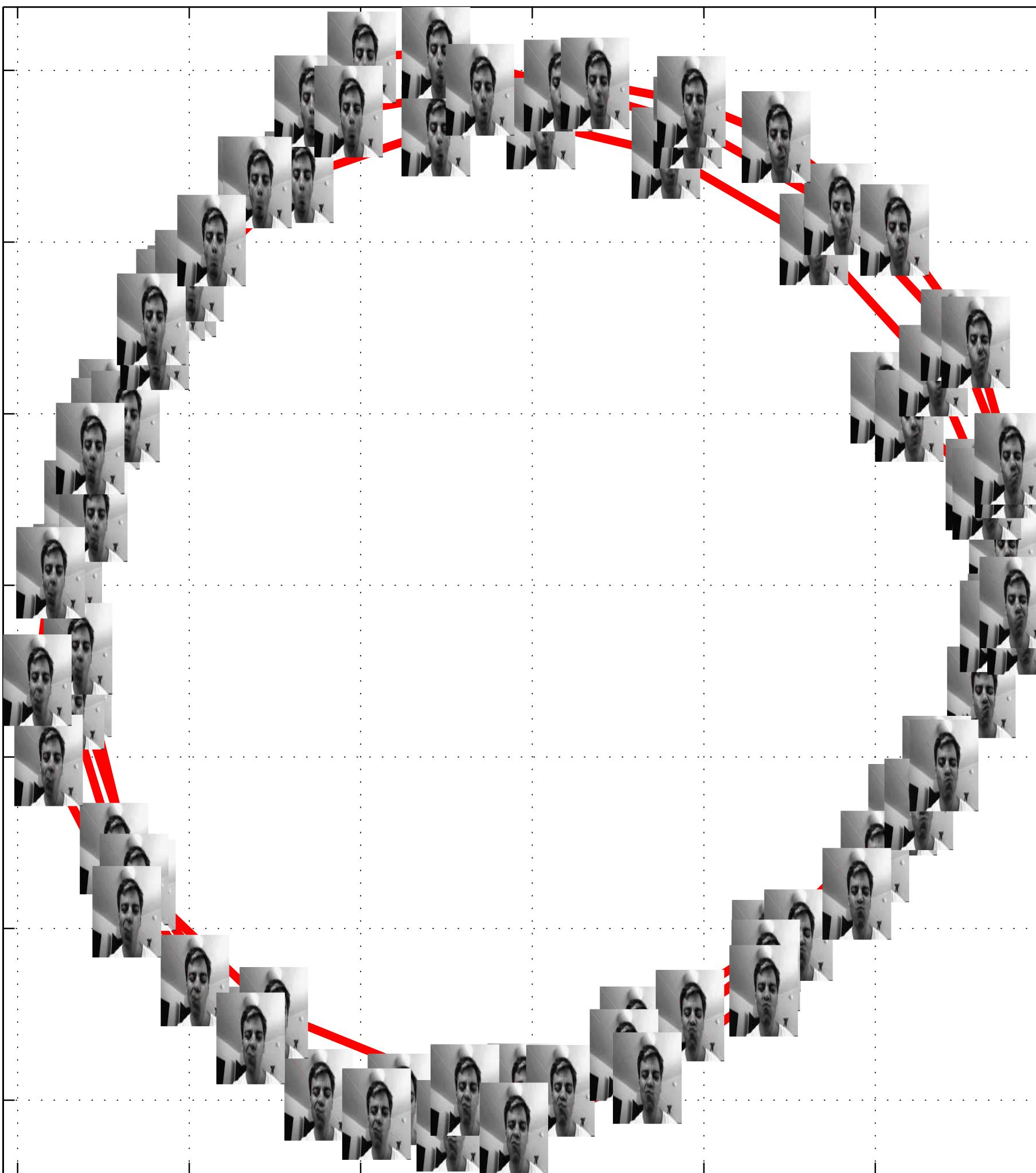


MDS

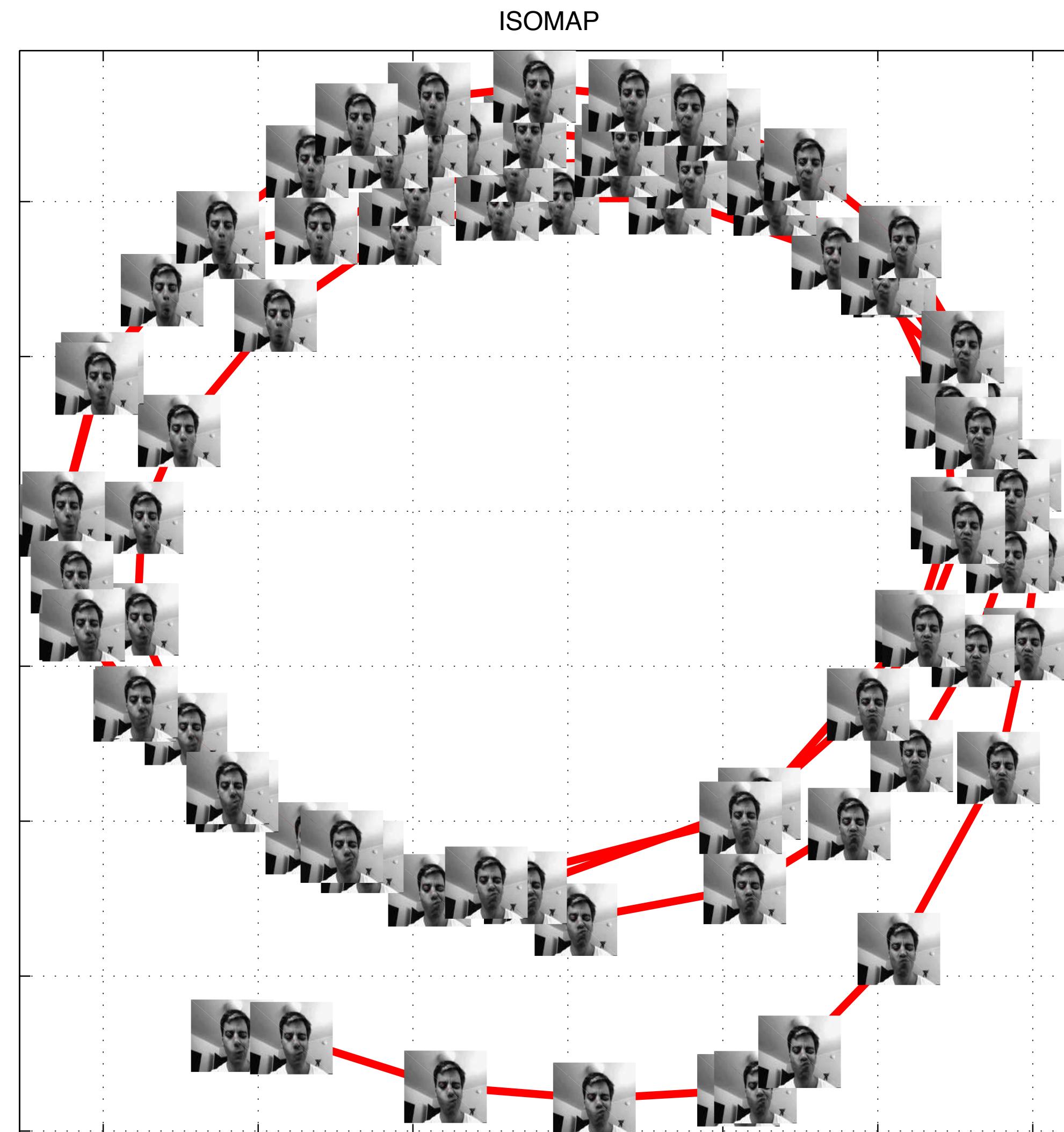


LLE

LLE



ISOMAP

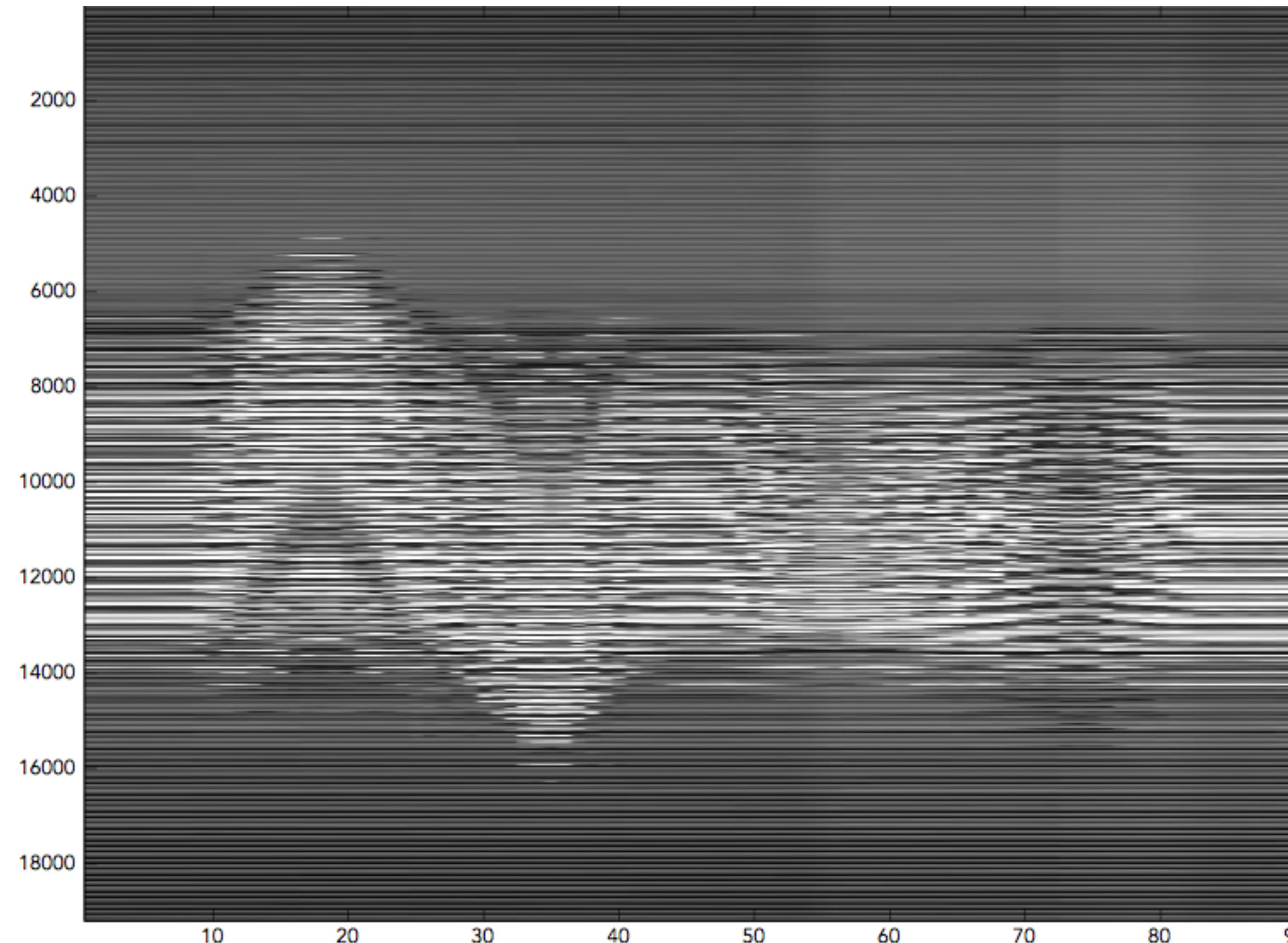


Another example movie

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

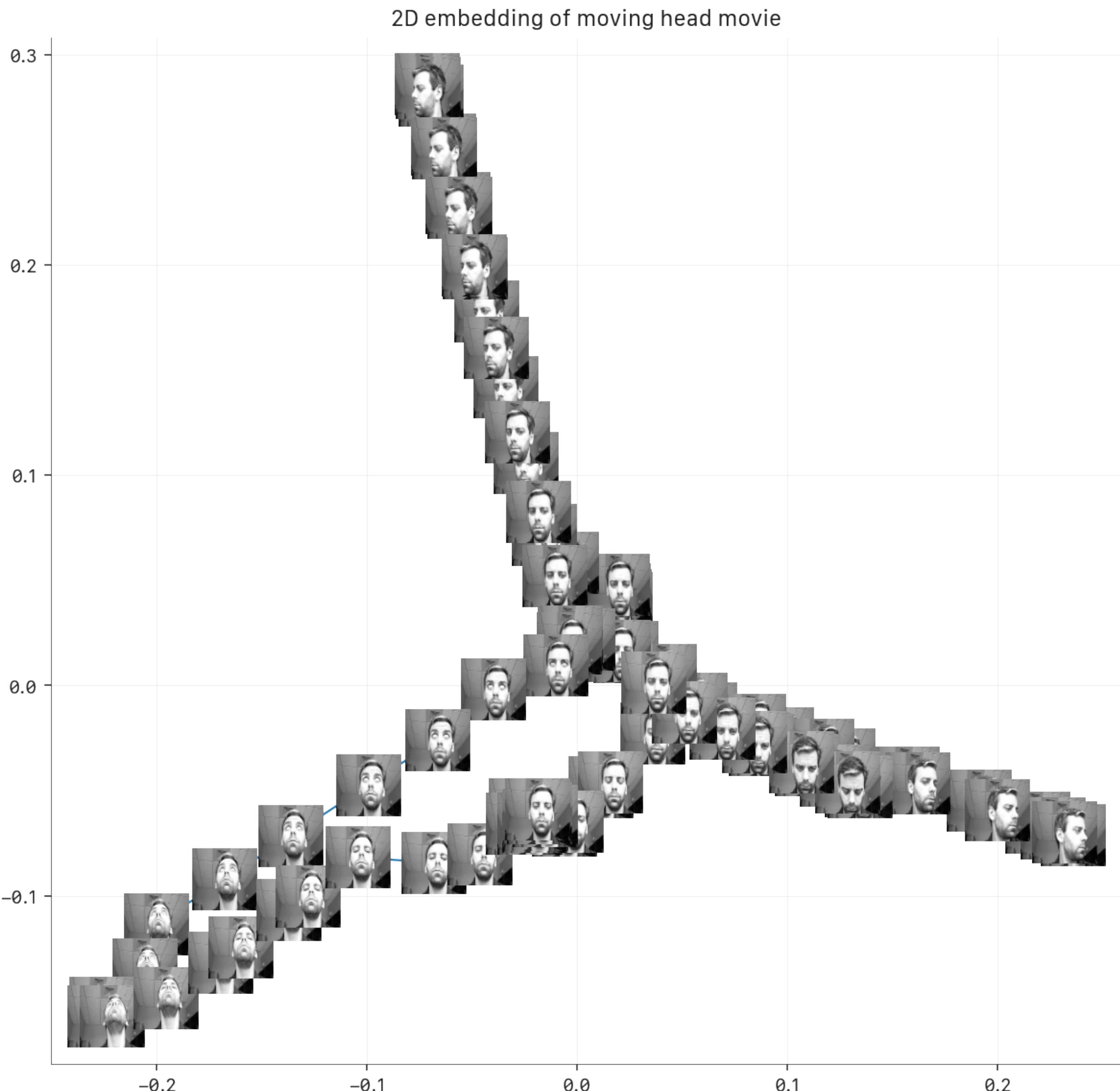


Input as a matrix



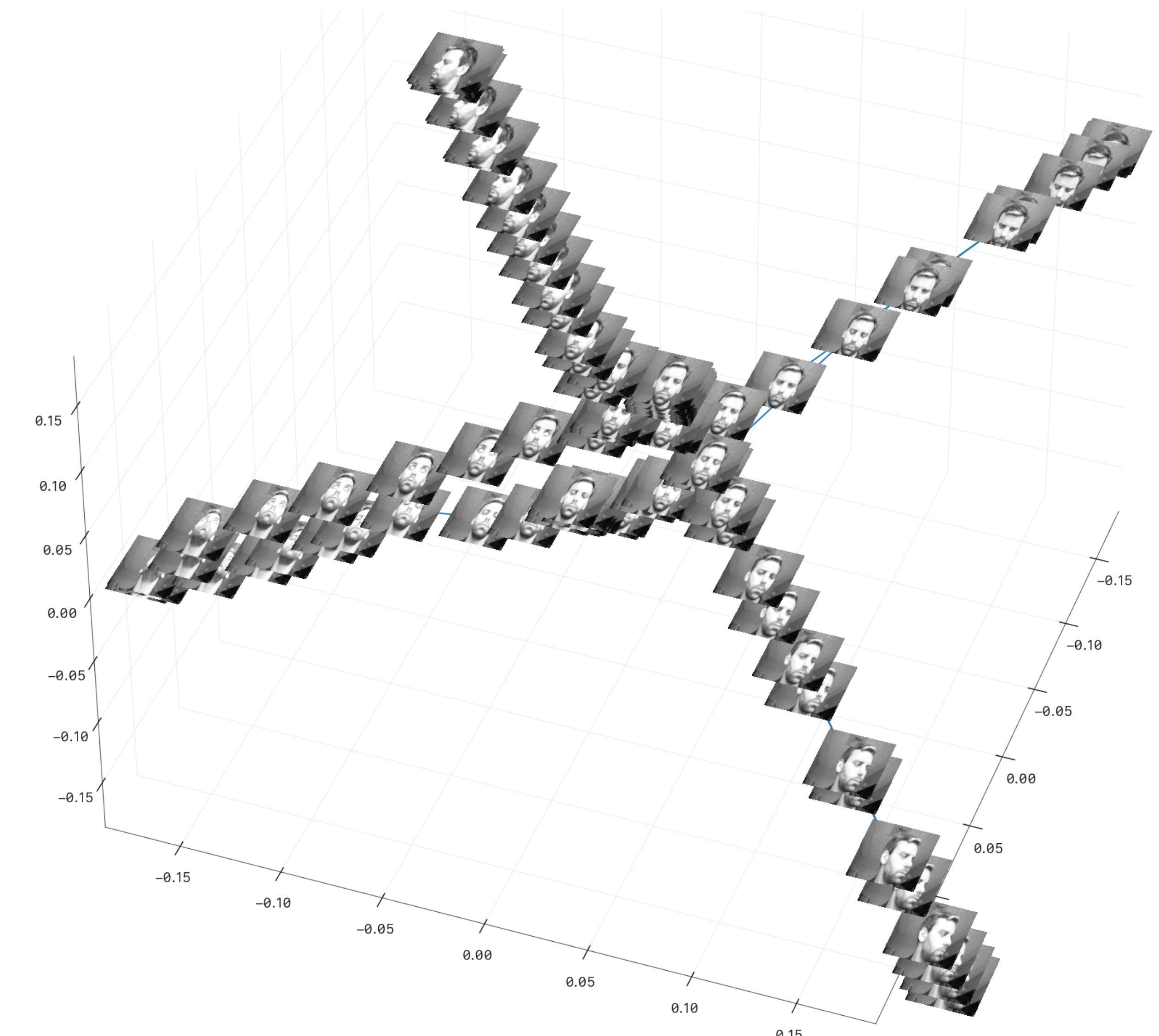
2D embedding of face video

- Stems coincide with head movements
- Not quite right
 - Two movements have been mapped on the same direction



3D embedding of face video

- Better!
- Extra dimension helps to separate movements



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

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://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=476974F6AA53BD038615E67656102714?doi=10.1.1.128.7613&rep=rep1&type=pdf>
- Manifolds (optional)
 - <https://www.science.org/doi/10.1126/science.290.5500.2268>
 - <https://cs.nyu.edu/~roweis/lle/papers/lleintro.pdf>

And we'll have a new problem set out tonight!

- Less math, more coding!
- Important notes:
 - All submissions must be a rendered Python notebook
 - Write your own code for all numerical routines
 - i.e. PCA, ICA, NMF, etc. (do not use sklearn or equivalents)
- Due date is Sept 27th (2 weeks from tomorrow)