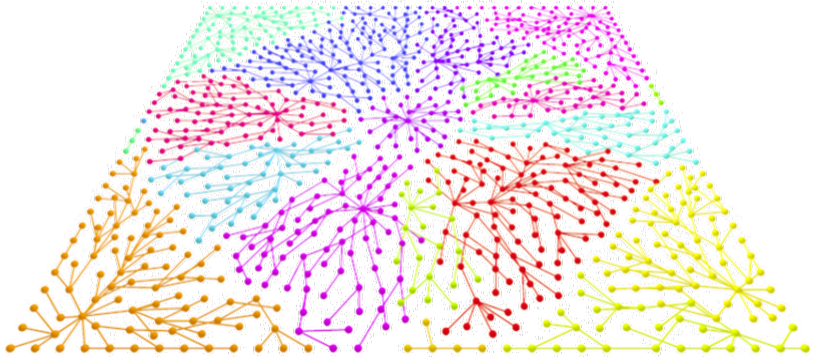
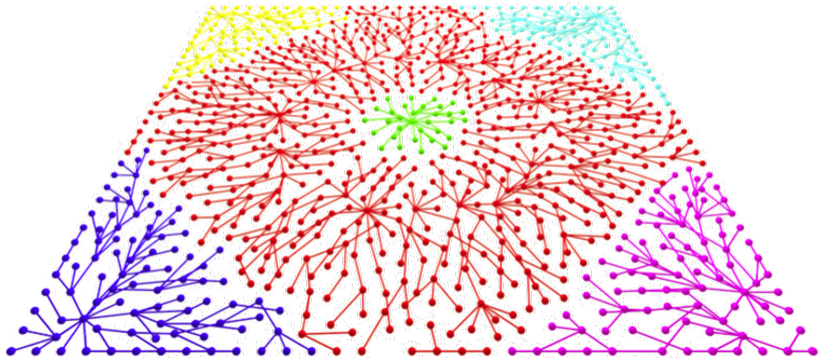
CS233, CME251: Geometric and Topological Data Analysis

Leonidas Guibas
Computer Science Department
Stanford University

Lecture 5
11 April 2022





Last Time:
CCA & MDS

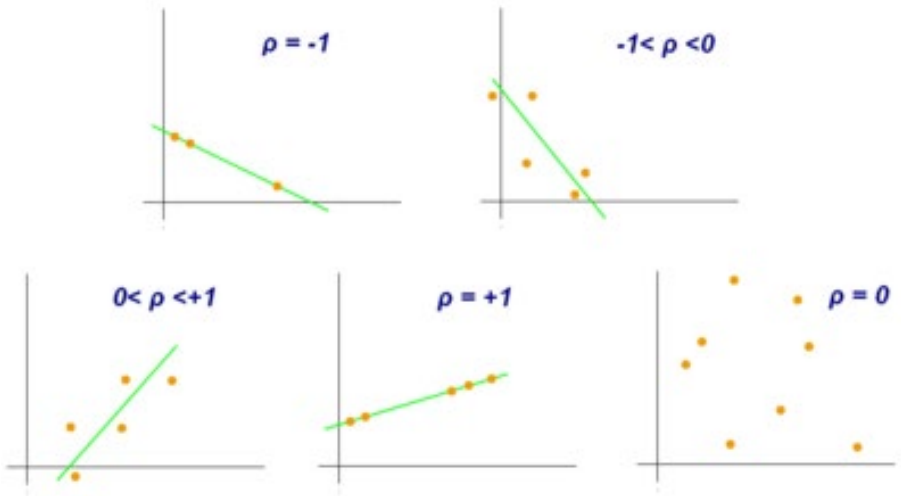
Canonical Correlation Analysis

Multi-dimensional Scaling

Covariance and Correlation

- Correlation measures a linear association between *X*, Y

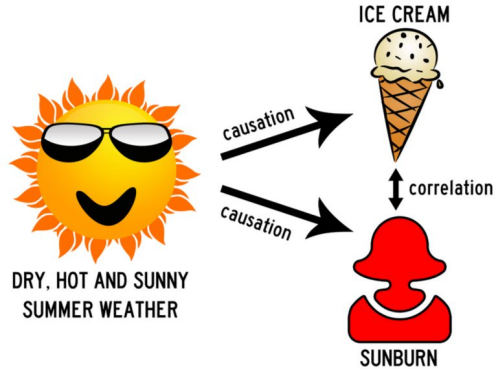
$$\rho = \mathrm{corr}(X, Y) = \frac{\mathrm{cov}(X, Y)}{\sqrt{\mathrm{var}(X)}\sqrt{\mathrm{var}(Y)}}$$



- Note that, *X*, Y independent, then corr(*X*,*Y*) = 0
  - but the opposite is not true

[*U* uniform in [0,2π], *X* = sin *U*, *Y* = cos *U*]

- High correlation not the same as causality



Canonical Correlations

- Canonical correlation analysis seeks a pair of linear transformations, one for each of the sets of variables *X*, *Y*, such that when the set of variables is transformed, the corresponding coordinates are maximally correlated.

$$S = ((\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n))$$

$$S_x = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$$
$$S_y = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$$

- Consider projections of *X* and *Y*

$$\begin{aligned} \boldsymbol{x} \rightarrow & \quad < \boldsymbol{w}_x, \boldsymbol{x} > \\ \boldsymbol{y} \rightarrow & \quad < \boldsymbol{w}_y, \boldsymbol{y} > \end{aligned}$$

- So we get

$$\boldsymbol{S}_{x,\boldsymbol{w}_x} = (< \boldsymbol{w}_x, \boldsymbol{x}_1 >, < \boldsymbol{w}_x, \boldsymbol{x}_2 >, \ldots, < \boldsymbol{w}_x, \boldsymbol{x}_n >)$$
$$\boldsymbol{S}_{y,\boldsymbol{w}_y} = (< \boldsymbol{w}_y, \boldsymbol{y}_1 >, < \boldsymbol{w}_y, \boldsymbol{y}_2 >, \ldots, < \boldsymbol{w}_y, \boldsymbol{y}_n >)$$

Covariance Formulation

$$\rho = \max_{\boldsymbol{w}_x, \boldsymbol{w}_y} \frac{E[< \boldsymbol{w}_x, \boldsymbol{x} >< \boldsymbol{w}_y, \boldsymbol{y} >]}{\sqrt{E[< \boldsymbol{w}_x, \boldsymbol{x} >^2] E[< \boldsymbol{w}_y, \boldsymbol{y} >^2]}}$$

- Can re-write as

$$\rho = \max_{\boldsymbol{w}_x, \boldsymbol{w}_y} \frac{E[\boldsymbol{w}_x^T \boldsymbol{x} \boldsymbol{y}^T \boldsymbol{w}_y]}{\sqrt{E[\boldsymbol{w}_x^T \boldsymbol{x} \boldsymbol{x}^T \boldsymbol{w}_{\boldsymbol{x}}] E[\boldsymbol{w}_y^T \boldsymbol{y} \boldsymbol{y}^T \boldsymbol{w}_{\boldsymbol{y}}]}}$$

- so that

$$\rho = \max_{\boldsymbol{w}_x, \boldsymbol{w}_y} \frac{\boldsymbol{w}_x^T E[\boldsymbol{x} \boldsymbol{y}^T] \boldsymbol{w}_y}{\sqrt{\boldsymbol{w}_x^T E[\boldsymbol{x} \boldsymbol{x}^T] \boldsymbol{w}_{\boldsymbol{x}} \boldsymbol{w}_y^T E[\boldsymbol{y} \boldsymbol{y}^T] \boldsymbol{w}_{\boldsymbol{y}}}}$$

- If we now write

$$C(\boldsymbol{x}, \boldsymbol{y}) = E\left[\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix}^T\right] = \begin{bmatrix} C_{\boldsymbol{x}\boldsymbol{x}} & C_{\boldsymbol{x}\boldsymbol{y}} \\ C_{\boldsymbol{y}\boldsymbol{x}} & C_{\boldsymbol{y}\boldsymbol{y}} \end{bmatrix} = C$$

- we get

$$\rho = \max_{\boldsymbol{w}_x, \boldsymbol{w}_y} \frac{\boldsymbol{w}_x^T C_{\boldsymbol{x}\boldsymbol{y}} \boldsymbol{w}_y}{\sqrt{\boldsymbol{w}_x^T C_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{w}_{\boldsymbol{x}} \boldsymbol{w}_y^T C_{\boldsymbol{y}\boldsymbol{y}} \boldsymbol{w}_{\boldsymbol{y}}}}$$

Constrained Optimization

- Note that the expression

$$\frac{\boldsymbol{w}_x^T C_{\boldsymbol{xy}} \boldsymbol{w}_y}{\sqrt{\boldsymbol{w}_x^T C_{\boldsymbol{xx}} \boldsymbol{w}_{\boldsymbol{x}} \boldsymbol{w}_y^T C_{\boldsymbol{yy}} \boldsymbol{w}_{\boldsymbol{y}}}}$$

  is invariant to re-scalings of w_x or w_y

- We can therefore solve the optimization problem

$$\max_{\boldsymbol{w}_x, \boldsymbol{w}_y} \quad \boldsymbol{w}_x^T C_{\boldsymbol{xy}} \boldsymbol{w}_y$$

  subject to the constraints

$$\boldsymbol{w}_x^T C_{\boldsymbol{xx}} \boldsymbol{w}_x = 1 \text{ and } \boldsymbol{w}_y^T C_{\boldsymbol{yy}} \boldsymbol{w}_y = 1.$$

CCA vs PCA

- Work with centered variates *X* and *Y* -- the covariance is $C_{\boldsymbol{xy}} = \boldsymbol{x}^T \boldsymbol{y}$
- Canonical correlation analysis attempts to answer the question "which directions account for most of the covariance between the two data sets?" The goal is to find directions w_x, w_y so as to maximize

$$\boldsymbol{w}_x^T C_{\boldsymbol{xy}} \boldsymbol{w}_y = (\boldsymbol{x}\boldsymbol{w}_x)^T (\boldsymbol{y}\boldsymbol{w}_y)$$

- subject to

$$\|\boldsymbol{x}\boldsymbol{w}_x\| = 1, \|\boldsymbol{y}\boldsymbol{w}_y\| = 1$$

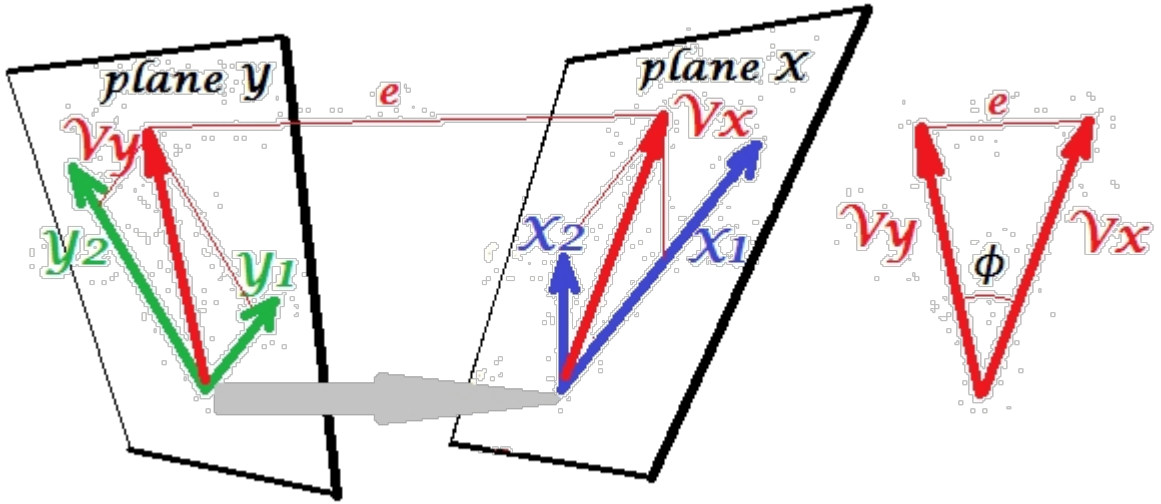
- In PCA we have a single variate *X* and seek the direction that "maximizes the variance in the data"

$$\boldsymbol{w}_x^T C_{\boldsymbol{xx}} \boldsymbol{w}_x = (\boldsymbol{x}\boldsymbol{w}_x)^T (\boldsymbol{x}\boldsymbol{w}_x)$$

- subject to

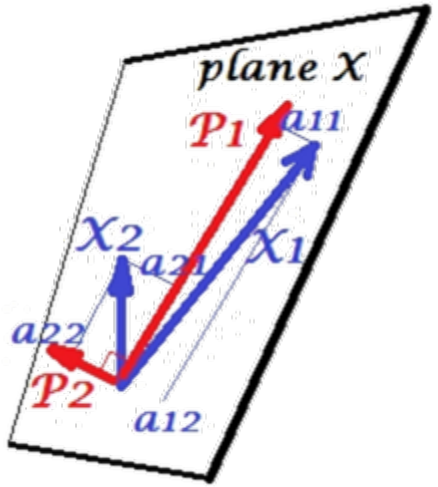
$$\|\boldsymbol{w}_x\| = 1$$

CCA vs PCA



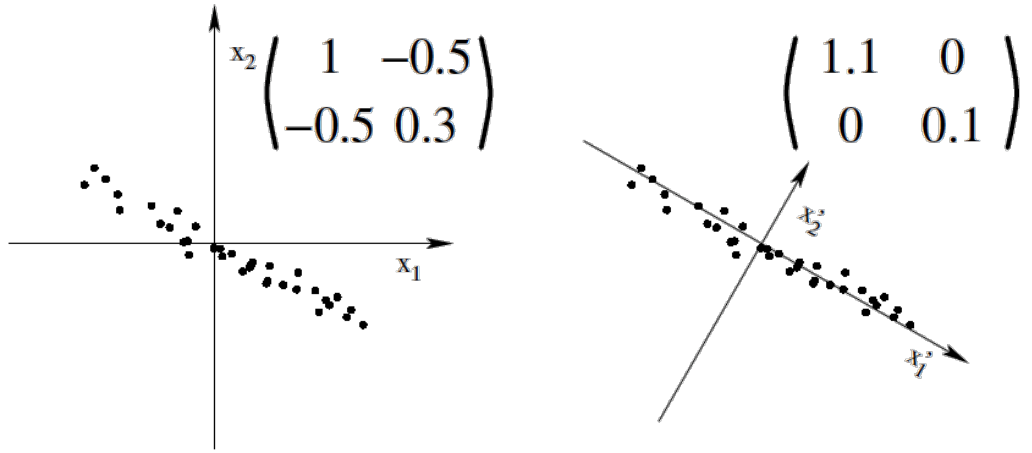
CCA

PCA



PCA by Diagonalizing the Covariance Matrix

- Finding the direction of maximum variance is easy if the covariance matrix is diagonal.



- So, in general, we want to rotate the coordinate system so as to make the covariance matrix diagonal.
- This is an eigenvalue problem; the eigenvectors of the covariance matrix point in the directions of maximal and minimal variance – so we rotate the axes to align them with the directions of maximal and minimal variance.

CCA (too) Becomes an Eigenvalue Problem

- A generalized eigenvalue problem

$$C_{\boldsymbol{xy}} C_{\boldsymbol{yy}}^{-1} C_{\boldsymbol{yx}} \boldsymbol{w}_x = \lambda^2 C_{\boldsymbol{xx}} \boldsymbol{w}_x$$

$$A\boldsymbol{x} = \lambda B\boldsymbol{x}$$

- Can symmetrically also get

$$C_{\boldsymbol{yx}} C_{\boldsymbol{xx}}^{-1} C_{\boldsymbol{xy}} \boldsymbol{w}_y = \lambda^2 C_{\boldsymbol{yy}} \boldsymbol{w}_y$$

- On can go back and forth between w_x and w_y

$$\boldsymbol{w}_y = \frac{C_{\boldsymbol{yy}}^{-1} C_{\boldsymbol{yx}} \boldsymbol{w}_x}{\lambda}$$

Higher-Order Canonical Correlates

- We defined the 1st canonical correlation ρ_1 through the projection vectors / directions

$$\boldsymbol{w}_x = \boldsymbol{w}_x^{(1)} \text{ and } \boldsymbol{w}_y = \boldsymbol{w}_y^{(1)}$$

- Given the first *k*-1 directions, the *k*-th canonical correlation is defined via vectors $\boldsymbol{w}_x^{(k)}$ and $\boldsymbol{w}_y^{(k)}$, so that we maximize

$$\max \quad (\boldsymbol{x}\boldsymbol{w}_x^{(k)})^T(\boldsymbol{y}\boldsymbol{w}_y^{(k)})$$

- but under orthogonality constraints to the previous directions

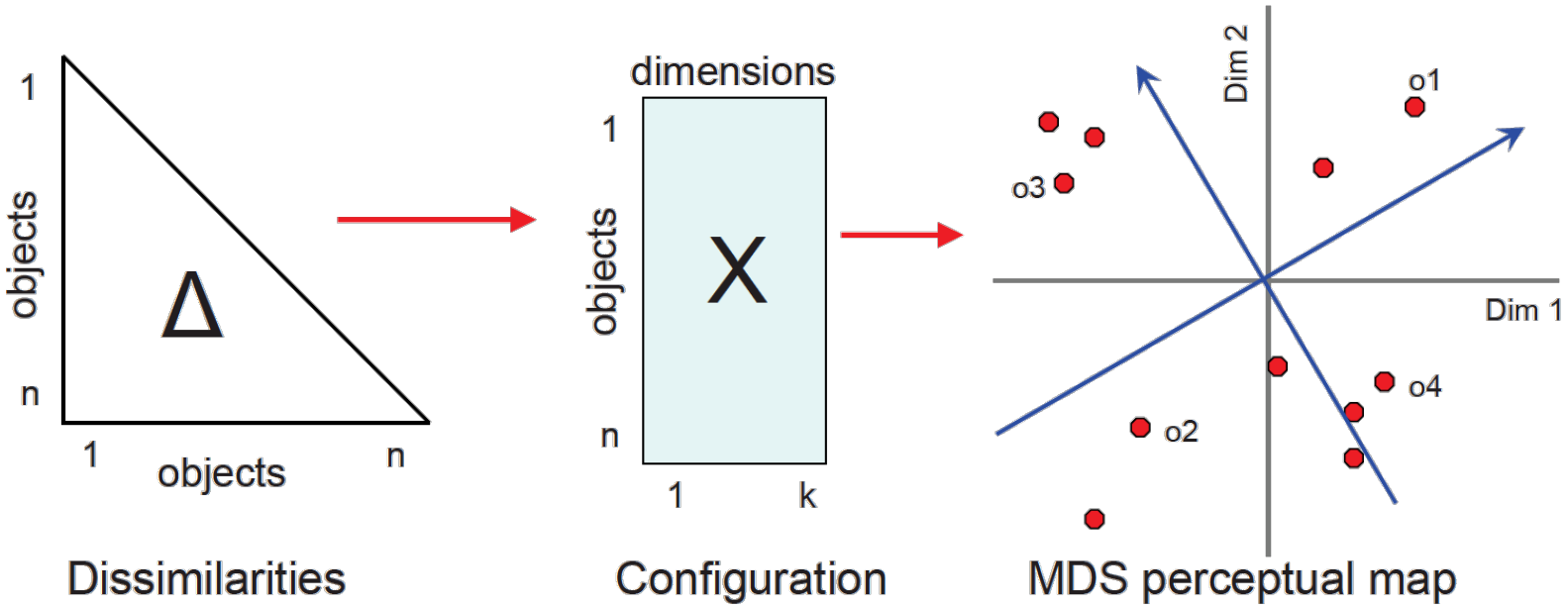
$$\|\boldsymbol{x}\boldsymbol{w}_x^{(k)}\| = 1, \|\boldsymbol{y}\boldsymbol{w}_y^{(k)}\| = 1$$

$$(\boldsymbol{x}\boldsymbol{w}_x^{(k)})^T(\boldsymbol{x}\boldsymbol{w}_x^{(j)}) = 0, j = 1, 2, \ldots, k-1$$

$$(\boldsymbol{y}\boldsymbol{w}_y^{(k)})^T(\boldsymbol{y}\boldsymbol{w}_y^{(j)}) = 0, j = 1, 2, \ldots, k-1$$

Multidimensional Scaling (MDS)

- A “distance preserving” embedding of the data into a Euclidean space
  - Sometimes distances are observed directly (e.g., similarity ratings)
  - Sometimes they can be calculated from a data table (e.g., Euclidean distances, correlations)



MDS: Motivating Example

- Travel times by train between French cities

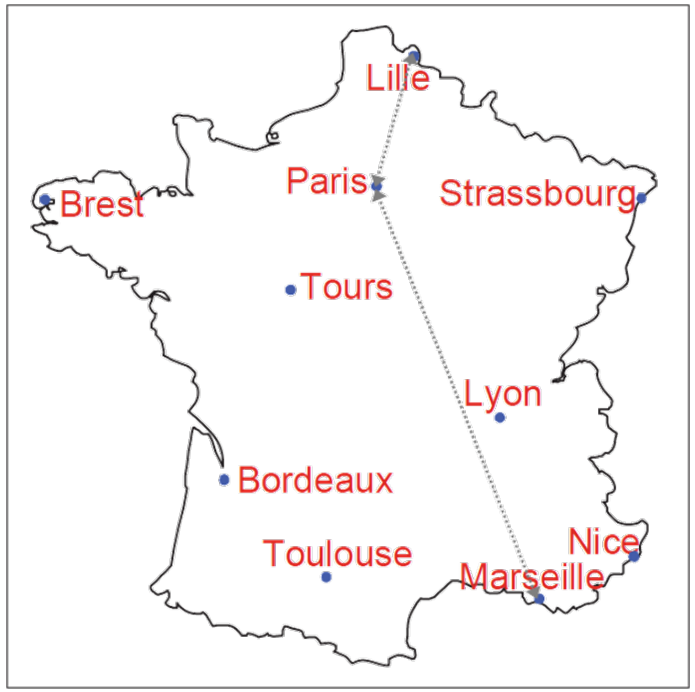
	Bor-deaux	Brest	Lille	Lyon	Mar-seille	Nice	Paris	Strassb ourg	Tou-louse	Tours
Bordeaux	0									
Brest	9:58	0								
Lille	6:39	7:11	0							
Lyon	8:05	7:11	4:52	0						
Marseille	5:47	8:49	6:12	1:35	0					
Nice	8:30	13:36	8:20	4:33	2:26	0				
Paris	2:59	4:17	1:04	2:01	3:00	5:52	0			
Strassbourg	8:08	10:16	6:54	4:36	7:04	11:15	4:01	0		
Toulouse	2:02	13:52	9:42	4:25	3:26	6:29	5:14	10:56	0	
Tours	2:36	5:38	4:17	4:21	5:13	9:04	1:13	6:03	6:06	0

- Considerations:
  - The recovered configuration (**X** = map) should be 2D
  - NSEW not relevant to distances– may have to rotate
  - Travel time not necessarily ~ map distance (TGV)
  - May need to consider other relations between dissimilarity and distance in MD space

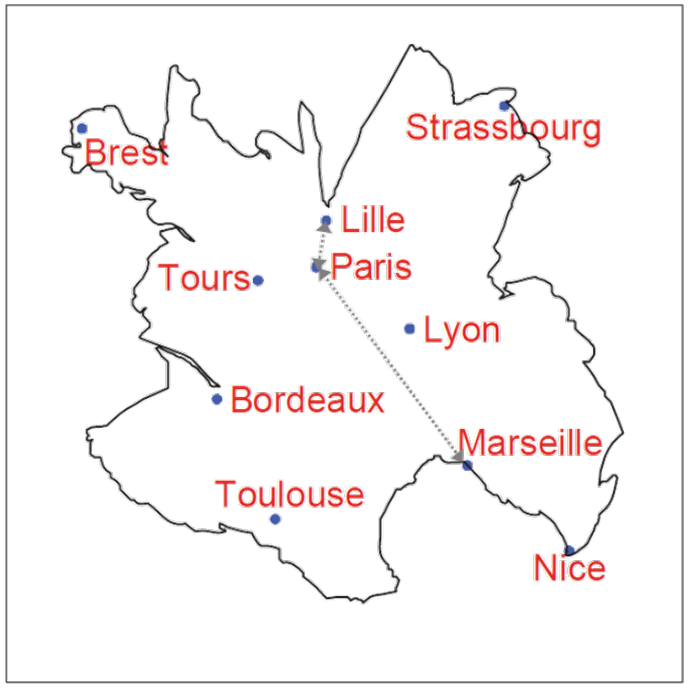
Source: Patrick Groenen, "Past, Present, and Future of Multidimensional Scaling", CARME, 2011

MDS: Motivating Example

- Travel times by train between French cities



Actual map of France



MDS recovered map

Marseille, Lille, Lyon, ... closer to Paris in travel time (TGV)

Formally (Metric MDS) …

- Given a (symmetric) matrix of pairwise "dis-similarities" between n objects / data sets

$$M = \left(\delta_{ij}\right)_{n\times n}$$

No need to satisfy the triangle inequality

- Find n points in low-dimensional space R^d, so that their distance matrix is as close as possible to M

- Low d (=2,3) allows us to visualize the data directly

Many MDS Variants

- Metric MDS

$$X'^T X' = B = \begin{pmatrix} | & & | \\ \mathbf{v_1} & \cdots & \mathbf{v_n} \\ | & & | \end{pmatrix} \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \begin{pmatrix} | & & | \\ \mathbf{v_1} & \cdots & \mathbf{v_n} \\ | & & | \end{pmatrix}^T$$

$$X'^T X' = \begin{pmatrix} | & & | \\ \mathbf{v_1} & \cdots & \mathbf{v_d} \\ | & & | \end{pmatrix} \begin{pmatrix} \mathbf{v}_n \end{pmatrix} \begin{pmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_d} \end{pmatrix} \begin{pmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_d} \end{pmatrix}^T \begin{pmatrix} | & & | \\ \mathbf{v_1} & \cdots & \mathbf{v_d} \\ | & & | \end{pmatrix}^T \begin{pmatrix} \mathbf{v}_n \end{pmatrix}^T$$

$d \times d$

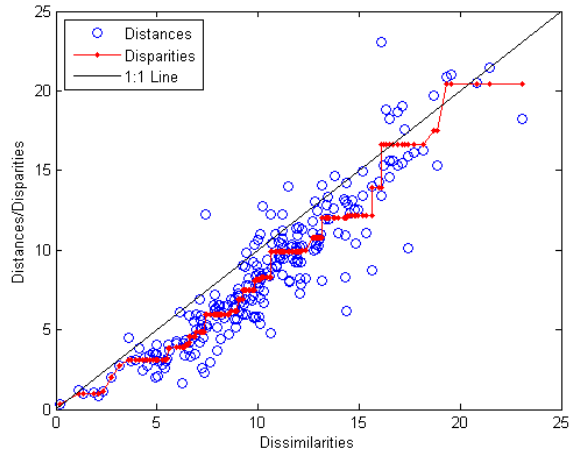
$n \times d$

X'^T $\quad$ X'

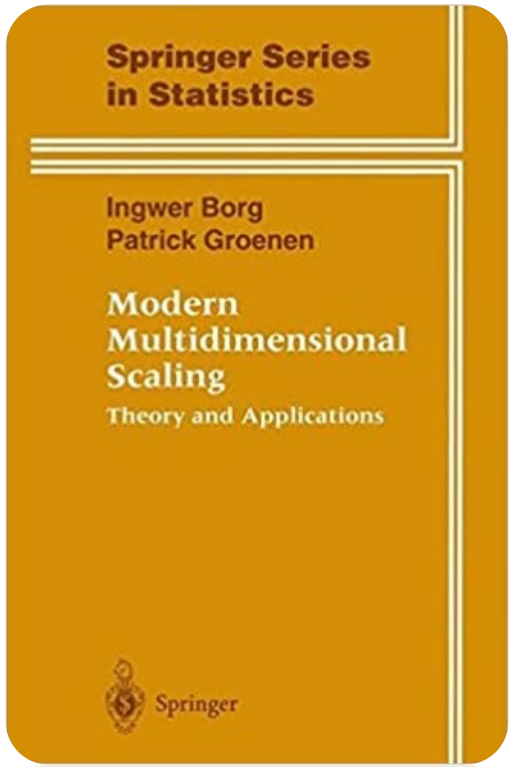
- Metric MDS with Sammon mapping

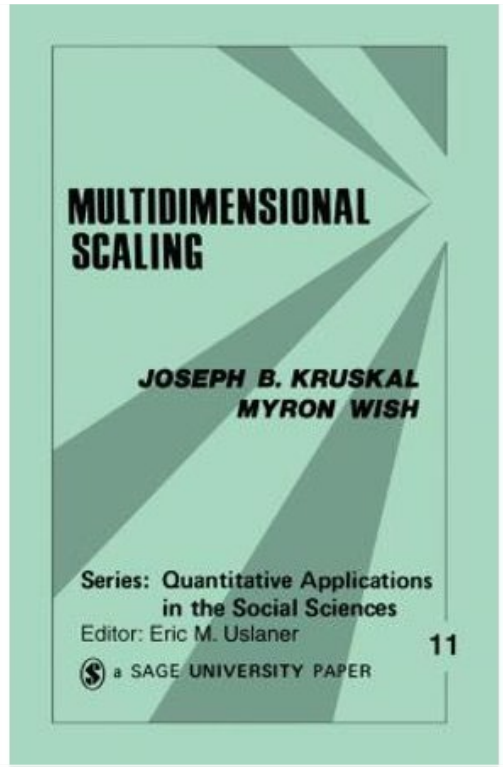
$$\text{stress} = \mathcal{L}(\hat{d}_{ij}) = \left(\sum_{i<j} (\hat{d}_{ij} - f(d_{ij}))^2 / \sum d_{ij}^2 \right)^{\frac{1}{2}}$$

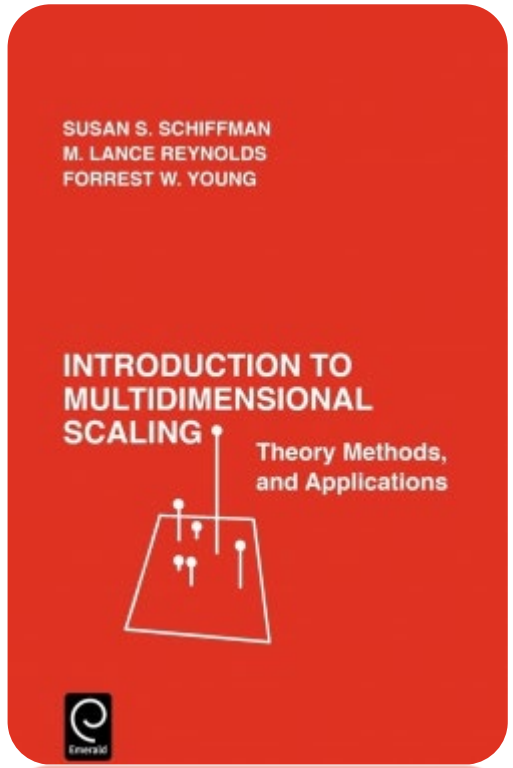
- Non-metric MDS



Many MDS Applications, Books, etc.

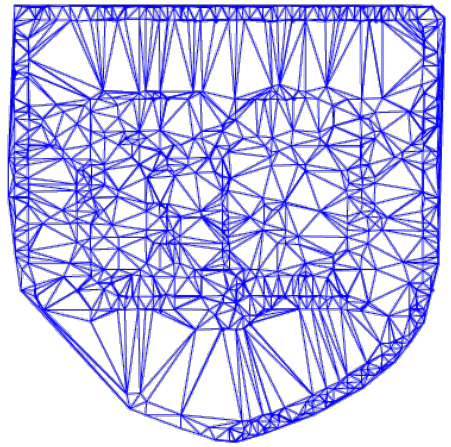


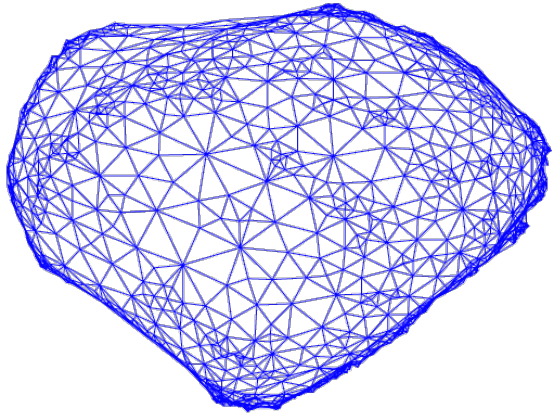




The Graph View of Data

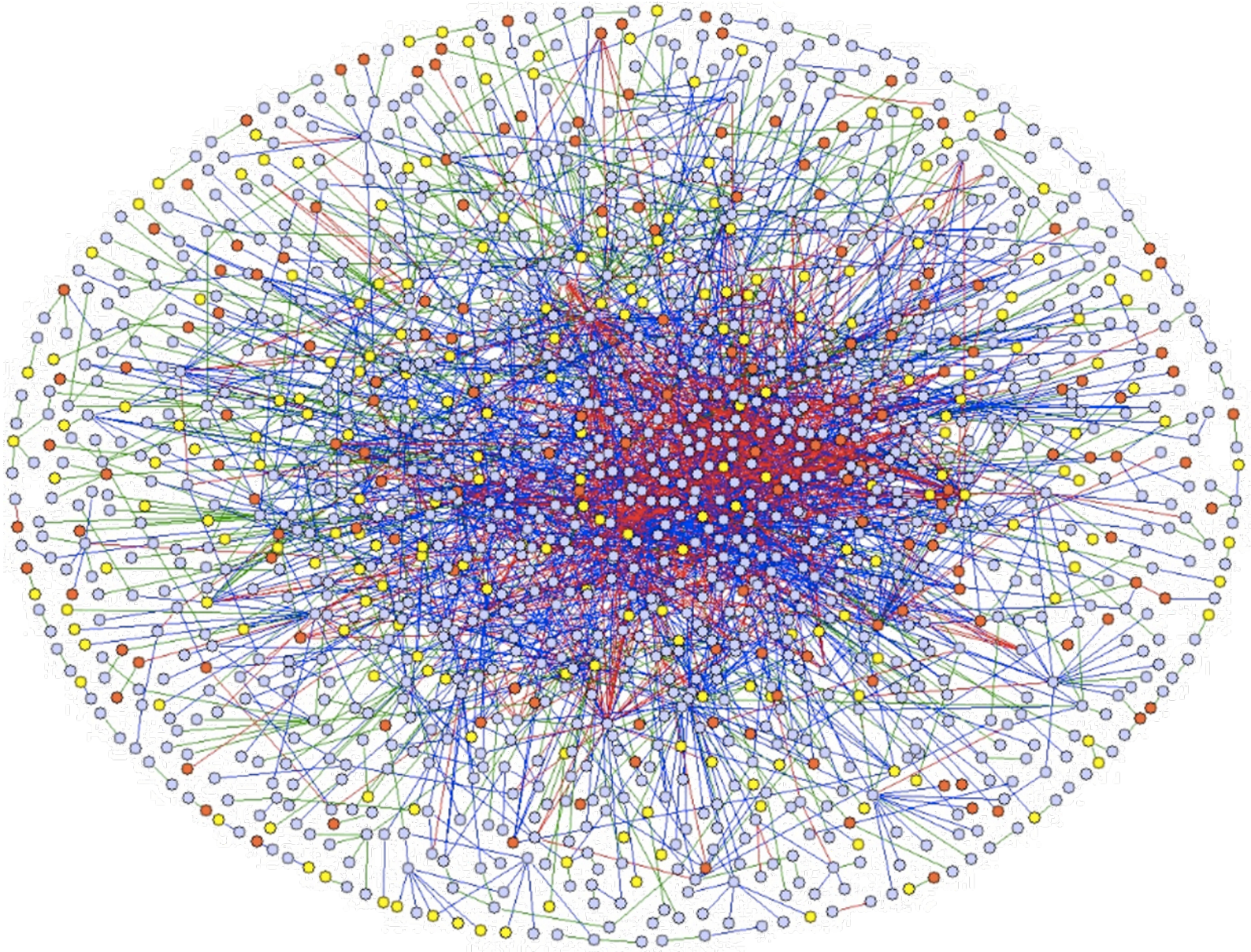
Graph Laplacians, Laplacian Embeddings, and Spectral Clustering





Slides ack: Radu Horaud, Dan Spielman
http://perception.inrialpes.fr/
http://www.cs.yale.edu/homes/spielman/561/

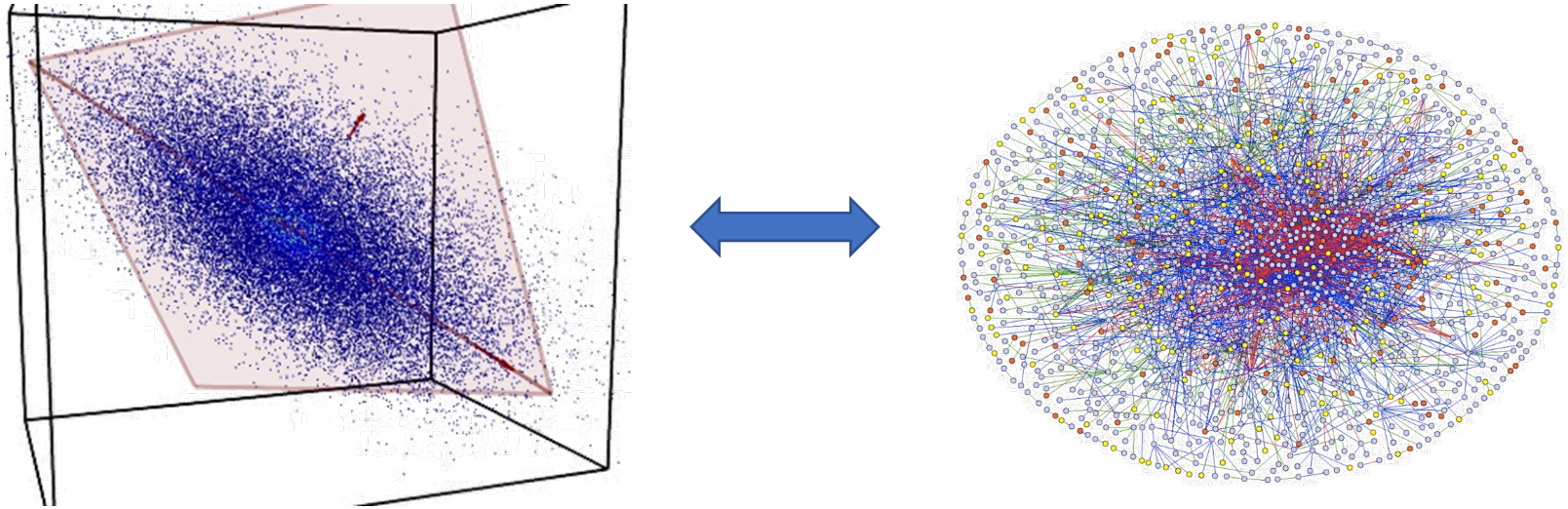
The Graph View of Data



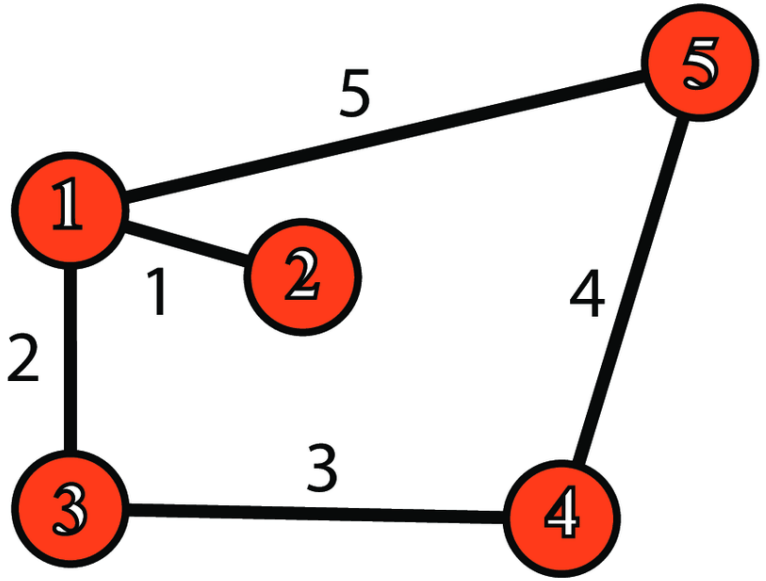
Social Networks

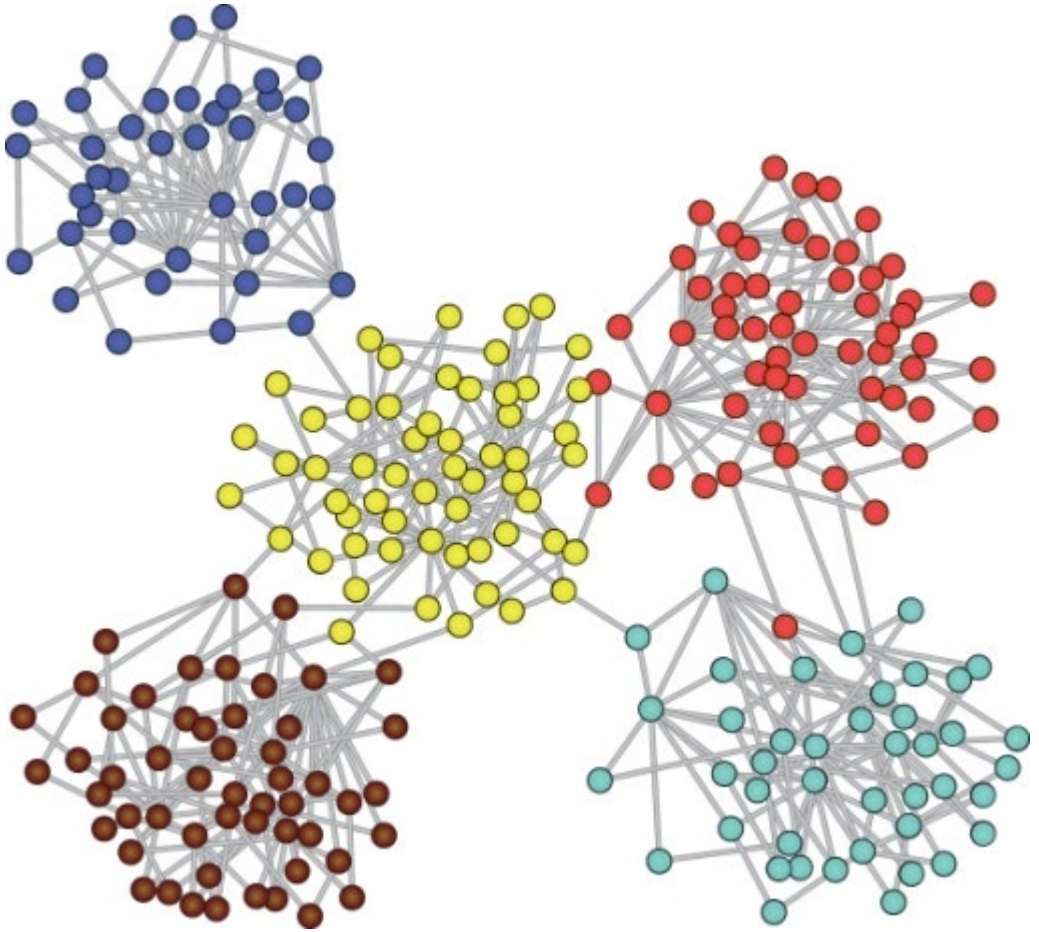
Connect Points in R^d and Graph Views of Data

- Euclidean -- points in R^d
  - via near-neighbor graphs
- Graph
  - via matrix representations of graphs



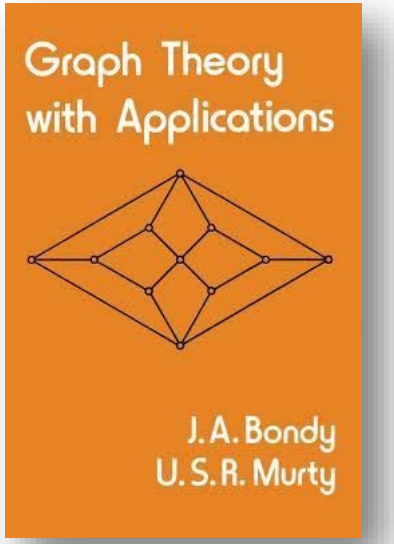
Graphs as Metric Spaces

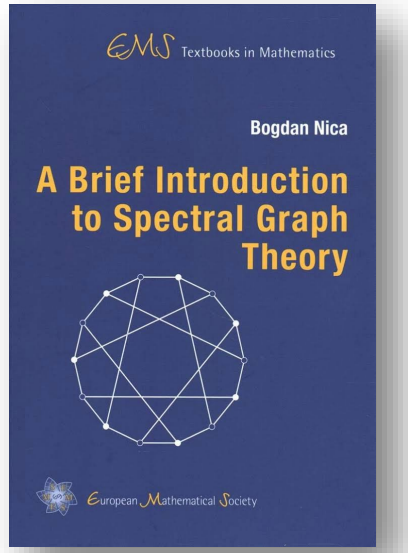




Spectral Graph Theory

Branch of Graph Theory





Spectral Graph Theory

- The *spectral graph theory* studies the properties of graphs via the eigenvalues and eigenvectors of their associated graph matrices: the *adjacency matrix* and the *graph Laplacian* and its variants.
- Both matrices have been extremely well studied from an algebraic point of view.
- The Laplacian allows a natural link between discrete representations, such as graphs, and continuous representations, such as vector spaces and manifolds.
- The most important application of the Laplacian is *spectral clustering* that corresponds to a computationally tractable solution to the *graph partitionning problem*.
- Another application is *spectral matching* that solves for *graph matching*.

Applications Across Many Fields

- *Spectral partitioning*: automatic circuit placement for VLSI (Alpert et al 1999), image segmentation (Shi & Malik 2000),
- *Text mining and web applications*: document classification based on semantic association of words (Lafon & Lee 2006), collaborative recommendation (Fouss et al. 2007), text categorization based on reader similarity (Kamvar et al. 2003).
- *Manifold analysis*: Manifold embedding, manifold learning, mesh segmentation, etc.

Graph Notations and Definitions

We consider *simple graphs* (no multiple edges or loops), $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$:

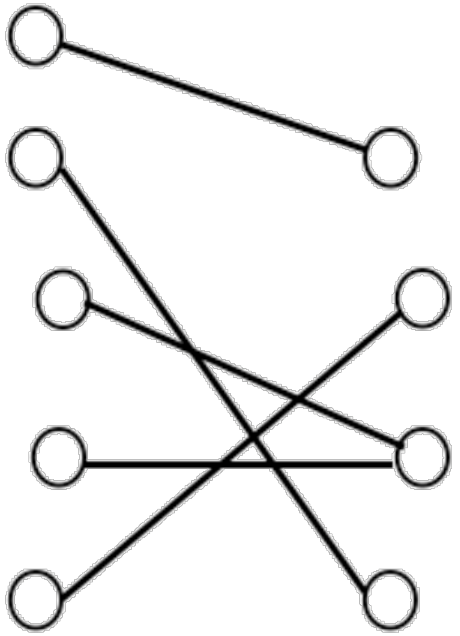
- $\mathcal{V}(\mathcal{G}) = \{v_1, \ldots, v_n\}$ is called the *vertex set* with $n = |\mathcal{V}|$;
- $\mathcal{E}(\mathcal{G}) = \{e_{ij}\}$ is called the *edge set* with $m = |\mathcal{E}|$;
- An edge e_{ij} connects vertices v_i and v_j if they are adjacent or neighbors. One possible notation for adjacency is $v_i \sim v_j$;
- The number of neighbors of a node v is called the *degree* of v and is denoted by $d(v)$, $d(v_i) = \sum_{v_i \sim v_j} e_{ij}$. If all the nodes of a graph have the same degree, the graph is *regular*; The nodes of an *Eulerian* graph have even degree.
- A graph is *complete* if there is an edge between every pair of vertices.

Subgraph Taxonomy

- $\mathcal{H}$ is a *subgraph* of $\mathcal{G}$ if $\mathcal{V}(\mathcal{H}) \subseteq \mathcal{V}(\mathcal{G})$ and $\mathcal{E}(\mathcal{H}) \subseteq \mathcal{E}(\mathcal{G})$;
- a subgraph $\mathcal{H}$ is an *induced subgraph* of $\mathcal{G}$ if two vertices of $\mathcal{V}(\mathcal{H})$ are adjacent if and only if they are adjacent in $\mathcal{G}$.
- A *clique* is a complete subgraph of a graph.
- A *path* of k vertices is a sequence of k distinct vertices such that consecutive vertices are adjacent.
- A *cycle* is a connected subgraph where every vertex has exactly two neighbors.
- A graph containing no cycles is a *forest*. A connected forest is a *tree*.

k-Partite Graphs

- A graph is called *k-partite* if its set of vertices admits a partition into k classes such that the vertices of the same class are not adjacent.
- An example of a *bipartite* graph.

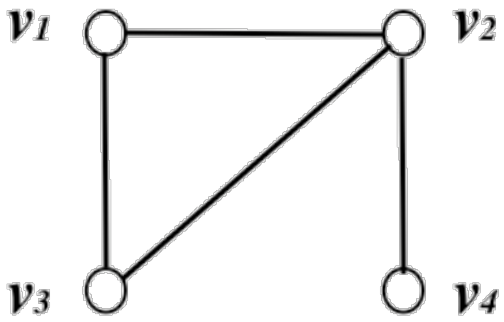


Adjacency Matrices

- For a graph with n vertices, the entries of the $n \times n$ adjacency matrix are defined by:

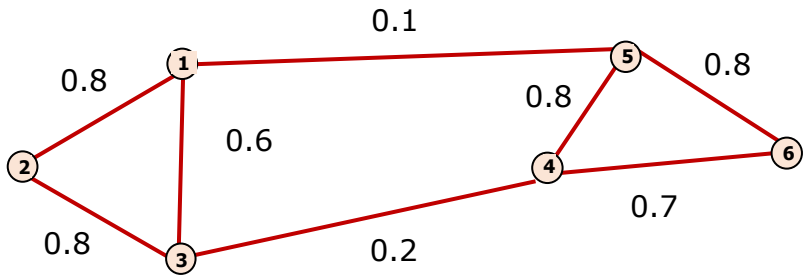
$$\mathbf{A} := \begin{cases} A_{ij} = 1 & \text{if there is an edge } e_{ij} \\ A_{ij} = 0 & \text{if there is no edge} \\ A_{ii} = 0 \end{cases}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$



Weighted Matrices

- Adjacency matrix (A)
  - $n \; x \; n$ matrix
  - $A = [w_{ij}]$ edge weight between vertex x_i and x_j



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0.8	0.6	0	0.1	0
x_2	0.8	0	0.8	0	0	0
x_3	0.6	0.8	0	0.2	0	0
x_4	0	0	0.2	0	0.8	0.7
x_5	0.1	0	0	0.8	0	0.8
x_6	0	0	0	0.7	0.8	0

- Important properties:
  - Symmetric matrix

  $\Rightarrow$ Eigenvalues are real

  $\Rightarrow$ Eigenvectors span orthogonal basis

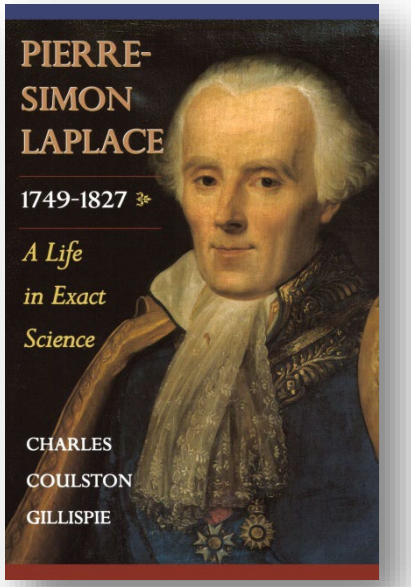
Eigenvalues and Eigenvectors

- $\mathbf{A}$ is a real-symmetric matrix: it has n real eigenvalues and its n real eigenvectors form an orthonormal basis.
- Let $\{\lambda_1, \ldots, \lambda_i, \ldots, \lambda_r\}$ be the set of *distinct* eigenvalues.
- The eigenspace S_i contains the eigenvectors associated with λ_i:

$$S_i = \{\boldsymbol{x} \in \mathbb{R}^n | \mathbf{A}\boldsymbol{x} = \lambda_i \boldsymbol{x}\}$$

- For real-symmetric matrices, the algebraic multiplicity is equal to the geometric multiplicity, for all the eigenvalues.
- The dimension of S_i (geometric multiplicity) is equal to the multiplicity of λ_i.
- If $\lambda_i \neq \lambda_j$ then S_i and S_j are mutually orthogonal.

Order the eigenvalues from small to large

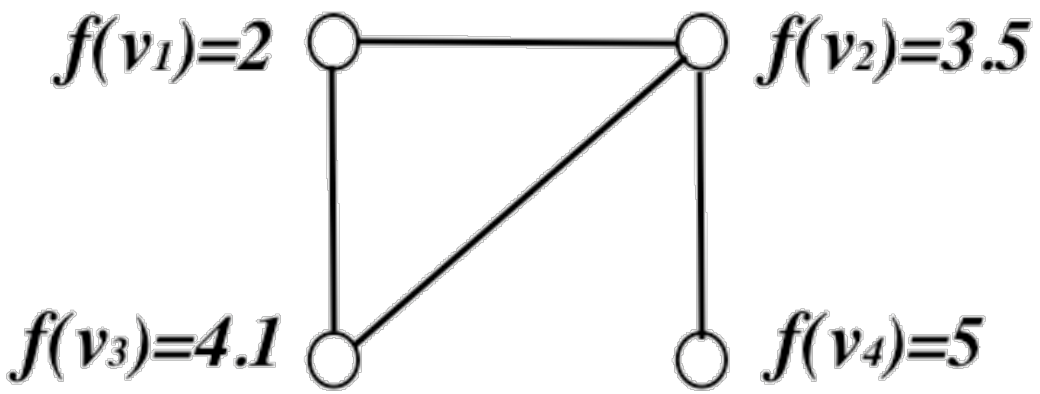


The Laplacian

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}$$

Functions on Graphs

- We consider real-valued functions on the set of the graph's vertices, $\boldsymbol{f} : \mathcal{V} \longrightarrow \mathbb{R}$. Such a function assigns a real number to each graph node.
- $\boldsymbol{f}$ is a vector indexed by the graph's vertices, hence $\boldsymbol{f} \in \mathbb{R}^n$.
- Notation: $\boldsymbol{f} = (f(v_1), \ldots, f(v_n)) = (f(1), \ldots, f(n))$.
- The eigenvectors of the adjacency matrix, $\mathbf{A}\boldsymbol{x} = \lambda\boldsymbol{x}$, can be viewed as *eigenfunctions*.



Operators and Quadratic Forms

- The adjacency matrix can be viewed as an operator

$$\boldsymbol{g} = \mathbf{A}\boldsymbol{f}; g(i) = \sum_{i \sim j} f(j)$$

- It can also be viewed as a quadratic form:

$$\boldsymbol{f}^\top \mathbf{A} \boldsymbol{f} = \sum_{e_{ij}} f(i) f(j)$$

Incidence Matrices for Directed Graphs

- Let each edge in the graph have an arbitrary but fixed orientation;
- The incidence matrix of a graph is a $|\mathcal{E}| \times |\mathcal{V}|$ $(m \times n)$ matrix defined as follows:

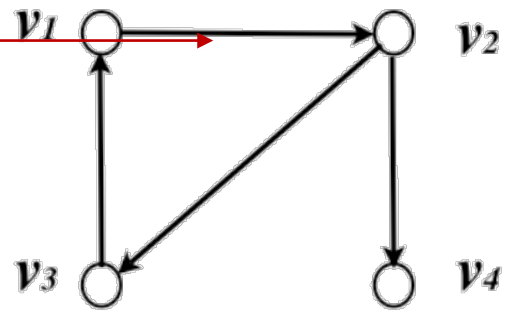
$$\nabla := \begin{cases} \nabla_{ev} = -1 & \text{if } v \text{ is the initial vertex of edge } e \\ \nabla_{ev} = 1 & \text{if } v \text{ is the terminal vertex of edge } e \\ \nabla_{ev} = 0 & \text{if } v \text{ is not in } e \end{cases}$$

Nabla

$$\nabla = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & +1 \end{bmatrix}$$

edges

vertices



Discrete Differential Operator

- The mapping $\boldsymbol{f} \longrightarrow \bigtriangledown \boldsymbol{f}$ is known as the *co-boundary mapping* of the graph.
- $(\bigtriangledown \boldsymbol{f})(e_{ij}) = f(v_j) - f(v_i)$

$$\begin{pmatrix} f(2) - f(1) \\ f(1) - f(3) \\ f(3) - f(2) \\ f(4) - f(2) \end{pmatrix} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & +1 \end{bmatrix} \begin{pmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \end{pmatrix}$$

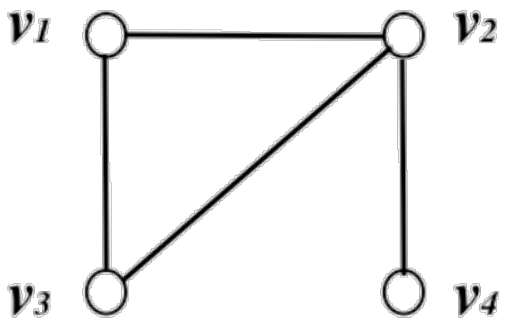
Graph (Unnormalized) Laplacian

- $\mathbf{L} = \nabla^\top \nabla$
- $(\mathbf{L}\boldsymbol{f})(v_i) = \sum_{v_j \sim v_i} (f(v_i) - f(v_j))$
- Connection between the Laplacian and the adjacency matrices:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

- The degree matrix: $\mathbf{D} := D_{ii} = d(v_i)$.

$$\mathbf{L} = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$



Laplacian Defines a Natural Quadratic Form of Graphs

$$x^T L x = \sum_{(i,j)\in E} (x(i) - x(j))^2$$

A measure of continuity of *x* over the graph

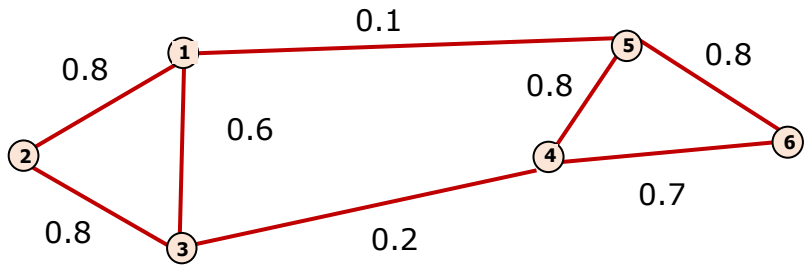
$$L = D - A$$ where D is diagonal matrix of degrees

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

Degree Matrix for Weighted Graphs

- **Degree matrix (D)**
  - $n \times n$ diagonal matrix
  - $D(i,i) = \sum_j w_{ij}$: total weight of edges incident to vertex x_i

$$D(i,i) = \sum_j w_{ij}$$



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.5	0	0	0	0	0
x_2	0	1.6	0	0	0	0
x_3	0	0	1.6	0	0	0
x_4	0	0	0	1.7	0	0
x_5	0	0	0	0	1.7	0
x_6	0	0	0	0	0	1.5

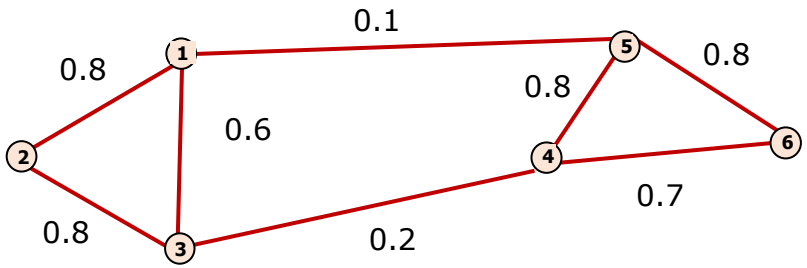
- Important application:
  - Normalize adjacency matrix

$$\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{1/2}$$

Laplacian Matrix for Weighted Graphs

- **Laplacian matrix (L)**
  - n x n symmetric matrix

$$L = D - A$$



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.5	-0.8	-0.6	0	-0.1	0
x_2	-0.8	1.6	-0.8	0	0	0
x_3	-0.6	-0.8	1.6	-0.2	0	0
x_4	0	0	-0.2	1.7	-0.8	-0.7
x_5	-0.1	0	0	-0.8	1.7	-0.8
x_6	0	0	0	-0.7	-0.8	1.5

- Important properties:
  - Eigenvalues are non-negative real numbers ← 0 always an eigenvalue
  - Eigenvectors are real and orthogonal
  - Eigenvalues and eigenvectors provide insight into the connectivity of the graph…

Undirected Weighted Graphs

- We consider *undirected weighted graphs*: Each edge e_{ij} is weighted by $w_{ij} > 0$.
- The Laplacian as an operator:

$$(\mathbf{L}\boldsymbol{f})(v_i) = \sum_{v_j \sim v_i} w_{ij}(f(v_i) - f(v_j))$$

- As a quadratic form:

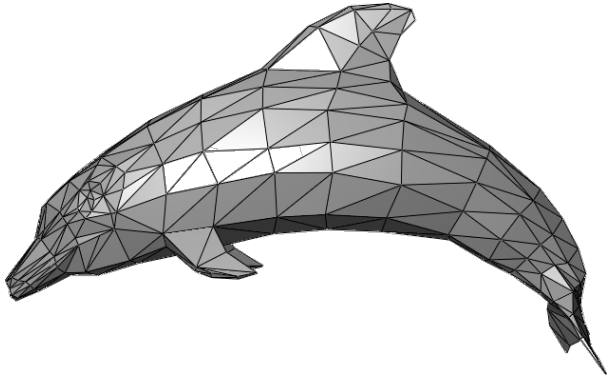
$$\boldsymbol{f}^\top \mathbf{L} \boldsymbol{f} = \sum_{e_{ij}} w_{ij}(f(v_i) - f(v_j))^2$$

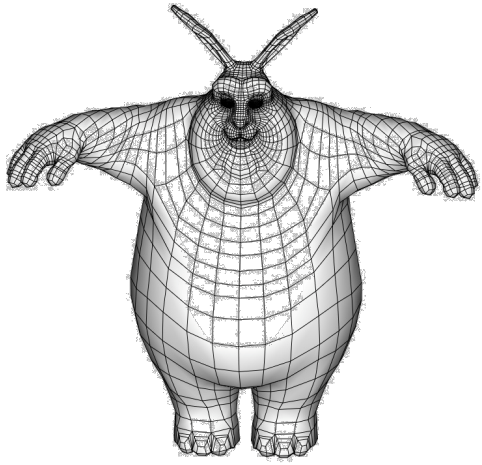
- $\mathbf{L}$ is symmetric and positive semi-definite.
- $\mathbf{L}$ has n non-negative, real-valued eigenvalues: $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$.

← 0 always an eigenvalue

Laplacians on Geometric Graphs

Discrete Surface Laplacians: 3D Meshes





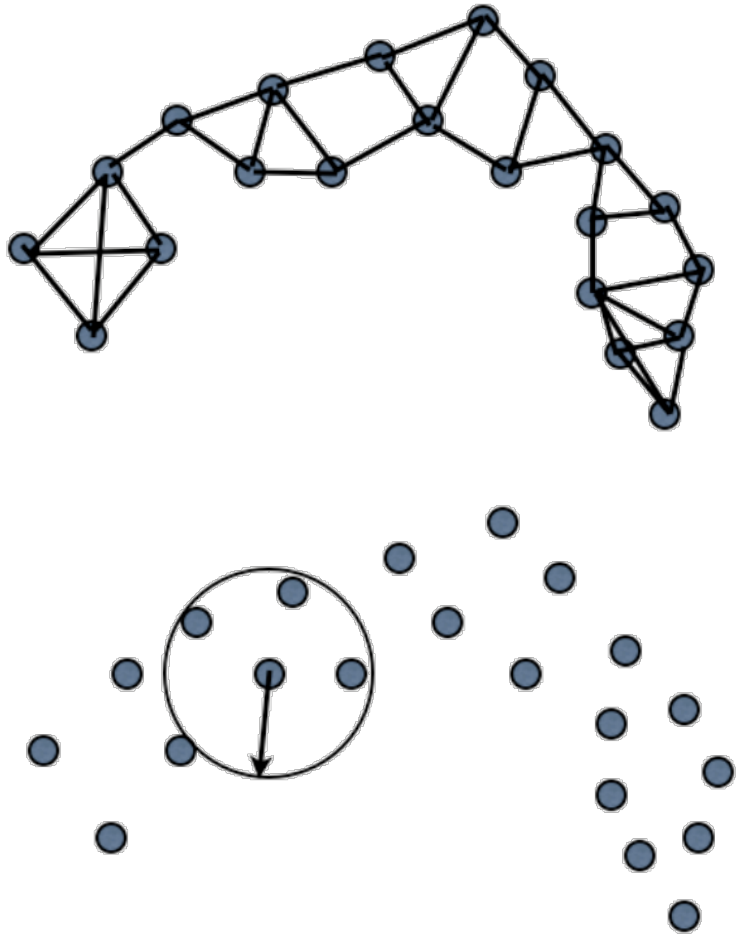
- A graph vertex v_i is associated with a 3D point $\boldsymbol{v}_i$.
- The weight of an edge e_{ij} is defined by the Gaussian kernel:

$$w_{ij} = \exp\left(-\|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2/\sigma^2\right)$$

- $0 \leq w_{\min} \leq w_{ij} \leq w_{\max} \leq 1$
- Hence, the geometric structure of the mesh is encoded in the weights.
- Other weighting functions were proposed in the literature.

Point Cloud Laplacians – Sparse Graphs

- 3-nearest neighbor graph
- ε-radius graph
- KNN may guarantee that the graph is connected (depends on the implementation)
- ε-radius does not guarantee that the graph has one connected component



Nearest Neighbor Graphs

Laplacians and Connectivity

Connected Graph Laplacians

$$w_{ij} \geq 0$$

- $\mathbf{L}\boldsymbol{u} = \lambda \boldsymbol{u}$.
- $\mathbf{L}\mathbf{1}_n = \mathbf{0}$, $\lambda_1 = 0$ is the smallest eigenvalue.
- The *one* vector: $\mathbf{1}_n = (1 \dots 1)^\top$.
- $0 = \boldsymbol{u}^\top \mathbf{L} \boldsymbol{u} = \sum_{i,j=1}^{n} w_{ij}(u(i) - u(j))^2$.
- If any two vertices are connected by a path, then $\boldsymbol{u} = (u(1), \dots, u(n))$ needs to be constant at all vertices such that the quadratic form vanishes. Therefore, a graph with one connected component has the constant vector $\boldsymbol{u}_1 = \mathbf{1}_n$ as the only eigenvector with eigenvalue 0.

A Graph with k Connected Components

- Each connected component has an associated Laplacian. Therefore, we can write matrix $\mathbf{L}$ as a *block diagonal matrix*:

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_1 & & \\ & \ddots & \\ & & \mathbf{L}_k \end{bmatrix}$$

- The spectrum of $\mathbf{L}$ is given by the union of the spectra of $\mathbf{L}_i$.
- Each block corresponds to a connected component, hence each matrix $\mathbf{L}_i$ has an eigenvalue 0 with multiplicity 1.
- The spectrum of $\mathbf{L}$ is given by the union of the spectra of $\mathbf{L}_i$.
- The eigenvalue $\lambda_1 = 0$ has multiplicity k.

The Eigenspace of $\lambda_1 = 0$

- The eigenspace corresponding to $\lambda_1 = \ldots = \lambda_k = 0$ is spanned by the k mutually orthogonal vectors:

$$\begin{aligned} \boldsymbol{u}_1 &= \mathbf{1}_{L_1} \\ &\ldots \\ \boldsymbol{u}_k &= \mathbf{1}_{L_k} \end{aligned}$$

- with $\mathbf{1}_{L_i} = (0000111110000)^\top \in \mathbb{R}^n$
- These vectors are the *indicator vectors* of the graph's connected components.
- Notice that $\mathbf{1}_{L_1} + \ldots + \mathbf{1}_{L_k} = \mathbf{1}_n$

Courant/Fischer Eigenvectors / Eigenvalues

$$\lambda_1 = \min_{x \neq 0} \frac{x^T L x}{x^T x} \qquad v_1 = \arg\min_{x \neq 0} \frac{x^T L x}{x^T x}$$

$$\lambda_2 = \min_{x \perp v_1} \frac{x^T L x}{x^T x} \qquad v_2 = \arg\min_{x \perp v_1} \frac{x^T L x}{x^T x}$$

(here $v_1 = \mathbf{1}$)

$$\lambda_k = \min_{S \text{ of dim } k} \max_{x \in S} \frac{x^T L x}{x^T x}$$

$$v_k = \arg \min_{x \perp v_1, \ldots, v_{k-1}} \frac{x^T L x}{x^T x}$$

The Fiedler Vector

- The first non-null eigenvalue λ_{k+1} is called the Fiedler value.
- The corresponding eigenvector $\boldsymbol{u}_{k+1}$ is called the Fiedler vector.
- The multiplicity of the Fiedler eigenvalue is always equal to 1.
- The Fiedler value is the *algebraic connectivity of a graph*, the further from 0, the more connected.
- The Fidler vector has been extensively used for *spectral bi-partioning*
- Theoretical results are summarized in Spielman & Teng 2007: `http://cs-www.cs.yale.edu/homes/spielman/`

Laplacian Eigenvectors for Connected Graphs

- $\boldsymbol{u}_1 = \mathbf{1}_n, \mathbf{L}\mathbf{1}_n = \mathbf{0}$.
- $\boldsymbol{u}_2$ is the *the Fiedler vector* with multiplicity 1.
- The eigenvectors form an orthonormal basis: $\boldsymbol{u}_i^\top \boldsymbol{u}_j = \delta_{ij}$.
- For any eigenvector $\boldsymbol{u}_i = (\boldsymbol{u}_i(v_1) \dots \boldsymbol{u}_i(v_n))^\top$, $2 \leq i \leq n$:

$$\boldsymbol{u}_i^\top \mathbf{1}_n = 0$$

- Hence the components of $\boldsymbol{u}_i$, $2 \leq i \leq n$ satisfy:

$$\sum_{j=1}^{n} \boldsymbol{u}_i(v_j) = 0$$

- Each component is bounded by:

$$-1 < \boldsymbol{u}_i(v_j) < 1$$

λ_2 = algebraic connectivity, monotone under graph inclusion
The larger λ_2, the more connected the graph.

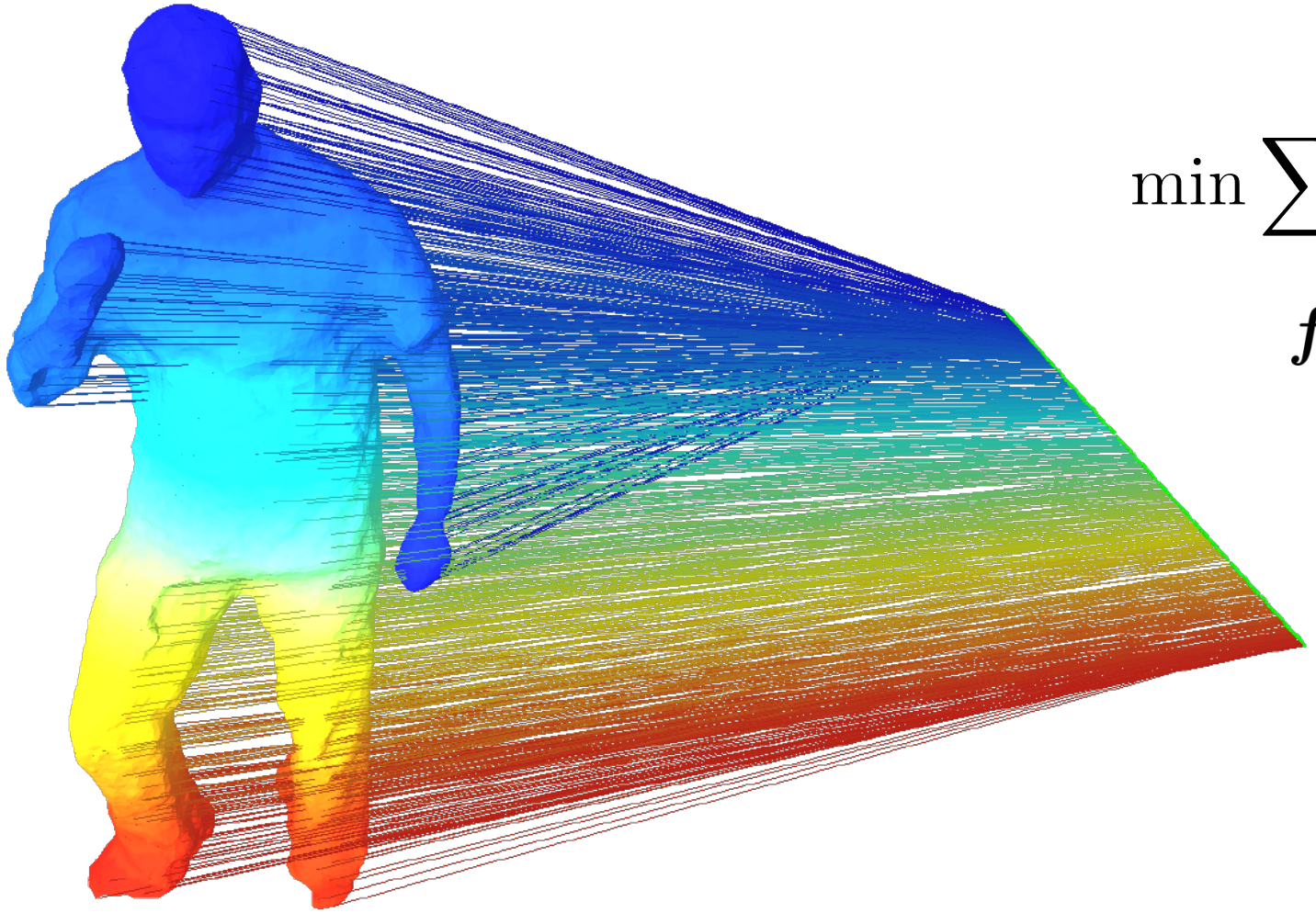
1-D Laplacian Embedding

- Map a weighted graph onto a line such that connected nodes stay as close as possible, i.e., minimize $\sum_{i,j=1}^{n} w_{ij}(f(v_i) - f(v_j))^2$, or:

$$\arg\min_{\boldsymbol{f}} \boldsymbol{f}^\top \mathbf{L}\boldsymbol{f} \text{ with: } \boldsymbol{f}^\top \boldsymbol{f} = 1 \text{ and } \boldsymbol{f}^\top \mathbf{1} = 0$$
$$\boldsymbol{f} \perp \mathbf{1}$$

- The solution is the eigenvector associated with the smallest nonzero eigenvalue of the eigenvalue problem: $\mathbf{L}\boldsymbol{f} = \lambda \boldsymbol{f}$, namely the Fiedler vector $\boldsymbol{u}_2$.
- For more details on this minimization see Golub & Van Loan *Matrix Computations*, chapter 8 (The symmetric eigenvalue problem).

1-D Embedding Example



$$\min \sum w_{ij}(f(v_i) - f(v_j))^2$$

$$\boldsymbol{f}^\top \boldsymbol{f} = 1 \quad \boldsymbol{f} \perp \mathbf{1}$$

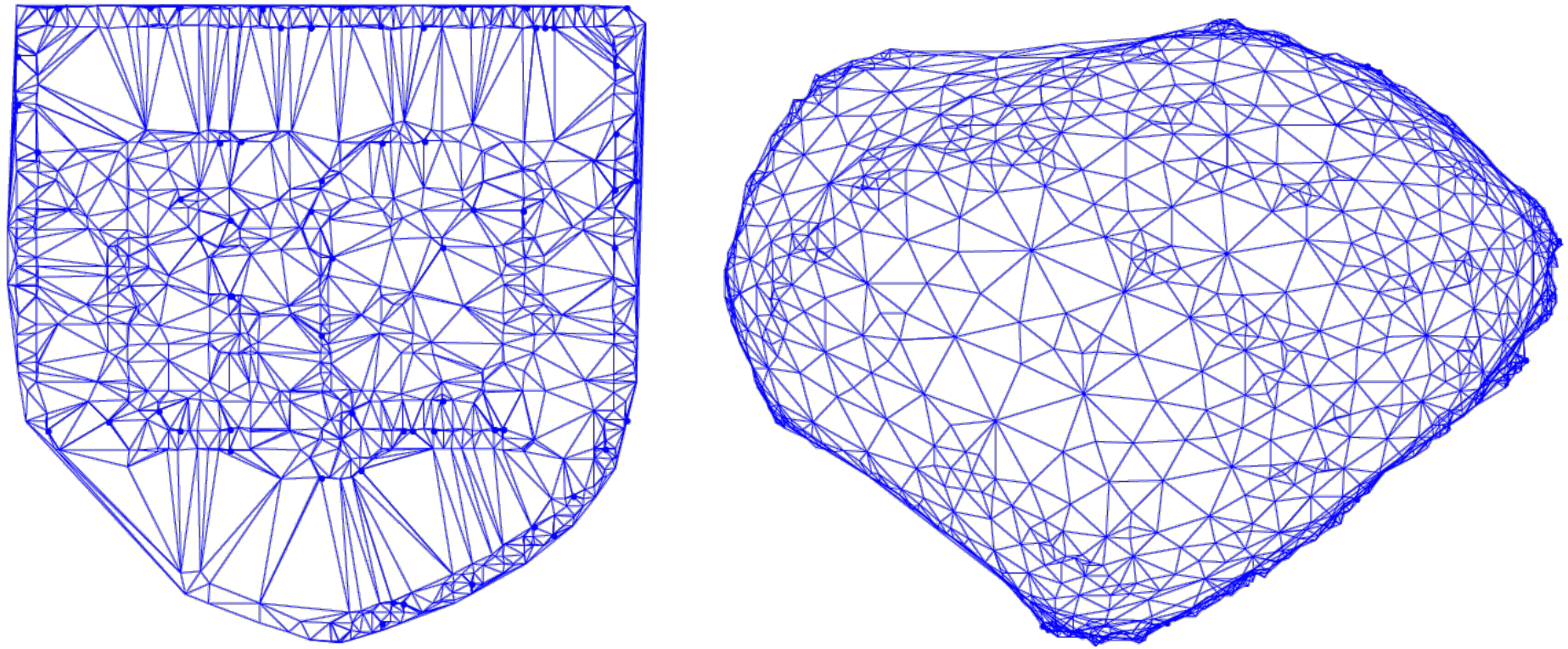
Higher-D Embeddings

- Embed the graph in a k-dimensional Euclidean space. The embedding is given by the $n \times k$ matrix $\mathbf{F} = [\boldsymbol{f}_1 \boldsymbol{f}_2 \dots \boldsymbol{f}_k]$ where the i-th row of this matrix – $\boldsymbol{f}^{(i)}$ – corresponds to the Euclidean coordinates of the i-th graph node v_i.
- We need to minimize (Belkin & Niyogi '03):

$$\arg \min_{\boldsymbol{f}_1 \cdots \boldsymbol{f}_k} \sum_{i,j=1}^{n} w_{ij} \|\boldsymbol{f}^{(i)} - \boldsymbol{f}^{(j)}\|^2 \text{ with: } \mathbf{F}^\top \mathbf{F} = \mathbf{I}.$$

- The solution is provided by the matrix of eigenvectors corresponding to the k lowest nonzero eigenvalues of the eigenvalue problem $\mathbf{L}\boldsymbol{f} = \lambda \boldsymbol{f}$.

Example: 2-D Embeddings



Spectral Graph Drawing

Condition for eigenvector $Lx = \lambda x$

Gives $x(i) = \frac{1}{d_i - \lambda} \sum_{j \sim i} x(j)$ for all i

λ small says x(i) near average of neighbors

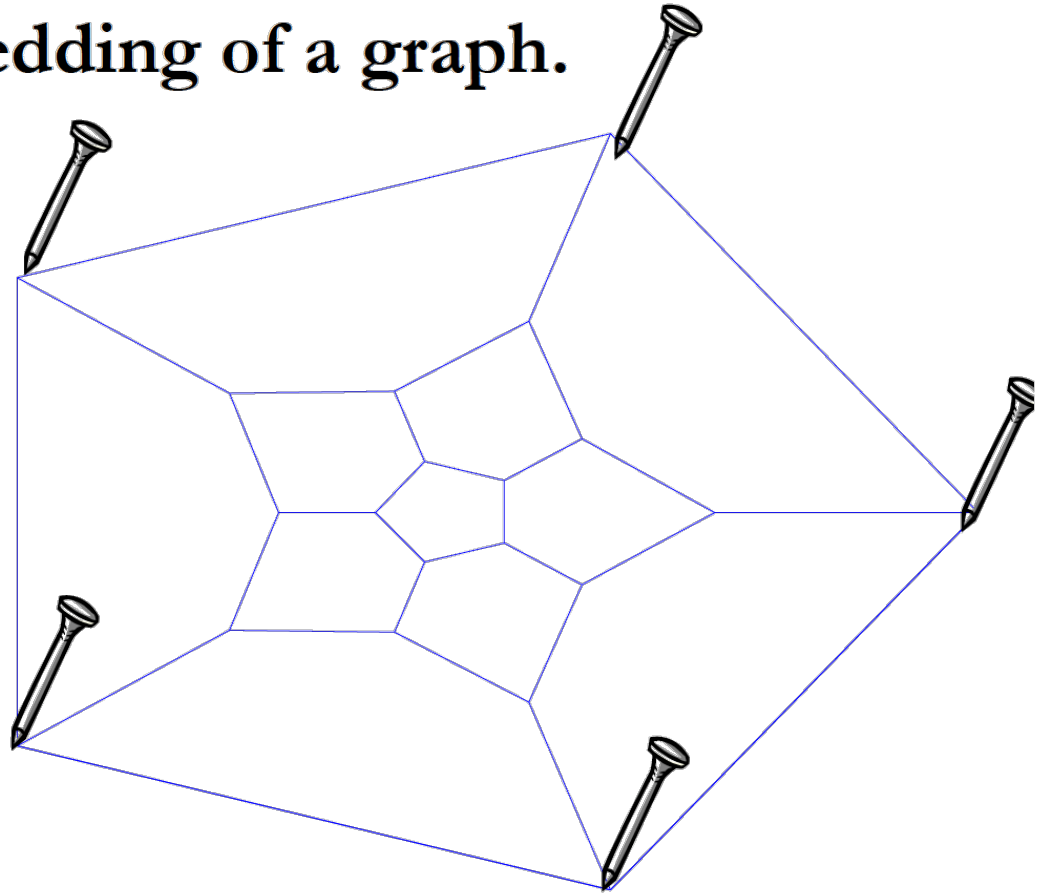
Tutte '63: If fix outside face, and let every other vertex be average of neighbors, get planar embedding of planar graph.

Tutte Embedding

Tutte '63 embedding of a graph.

Fix outside face.
Edges -> springs.

Vertex at center
of mass of nbrs.



William T. Tutte.

3-connected -> get planar embedding

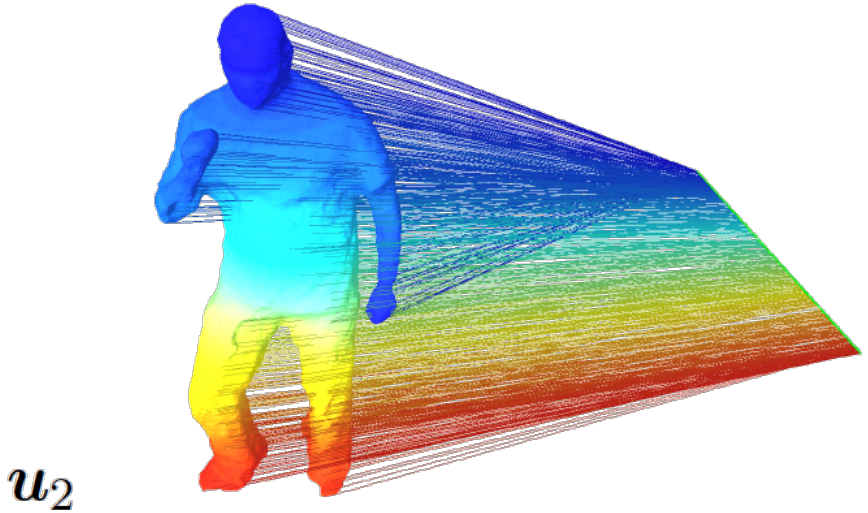
Spectral Embedding Using Unnormalized Laplacian

- Compute the eigendecomposition $\mathbf{L} = \mathbf{D} - \mathbf{A}$.
- Select the k smallest non-null eigenvalues $\lambda_2 \leq \ldots \leq \lambda_{k+1}$
- $\lambda_{k+2} - \lambda_{k+1} =$ **eigengap**.
- We obtain the $n \times k$ matrix $\mathbf{U} = [\boldsymbol{u}_2 \ldots \boldsymbol{u}_{k+1}]$:

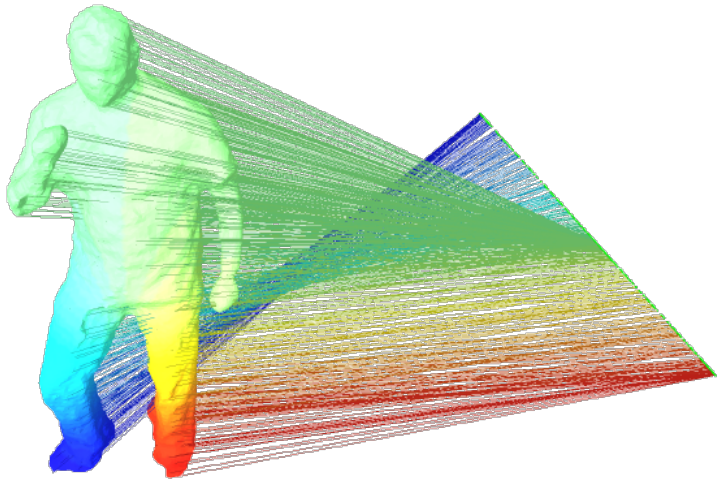
$$\mathbf{U} = \begin{bmatrix} \boldsymbol{u}_2(v_1) & \ldots & \boldsymbol{u}_{k+1}(v_1) \\ \vdots & & \vdots \\ \boldsymbol{u}_2(v_n) & \ldots & \boldsymbol{u}_{k+1}(v_n) \end{bmatrix}$$

- $\boldsymbol{u}_i^\top \boldsymbol{u}_j = \delta_{ij}$ (orthonormal vectors), hence $\mathbf{U}^\top \mathbf{U} = \mathbf{I}_k$.
- Column i $(2 \leq i \leq k+1)$ of this matrix is a mapping on the eigenvector $\boldsymbol{u}_i$.

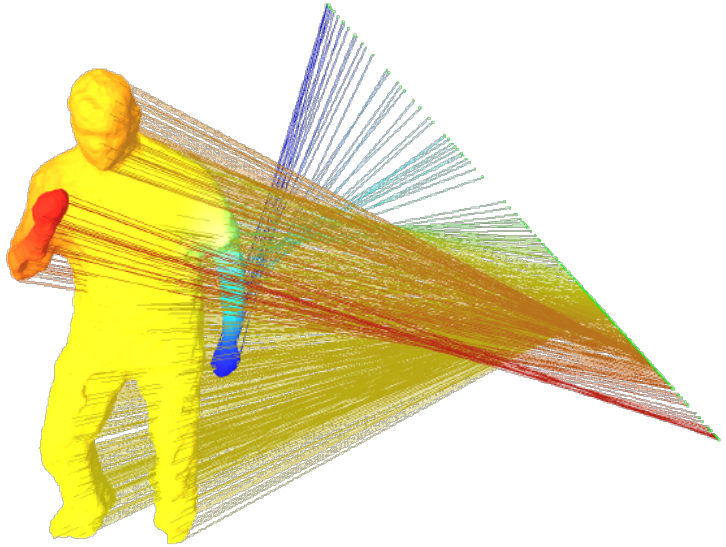
More Eigenvectors, More 1-D Embeddings



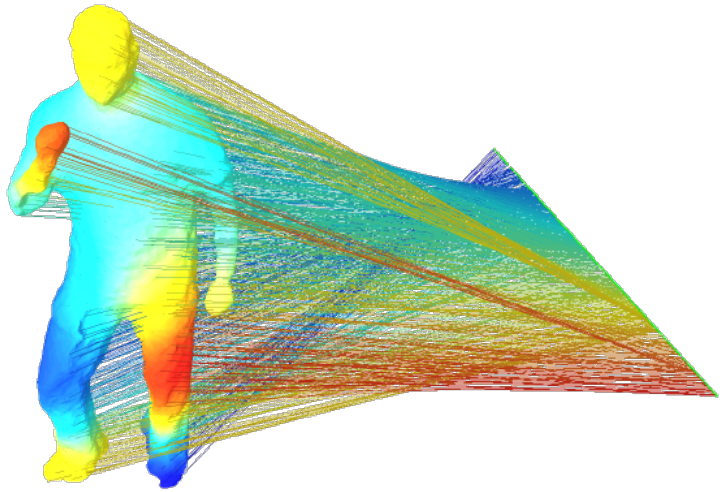
$\boldsymbol{u}_2$



$\boldsymbol{u}_3$



$\boldsymbol{u}_4$



$\boldsymbol{u}_8$

Laplacian Variants (Many)

The Normalized Spectral Embedding of a Graph

$$\mathbf{\Lambda}_k = \begin{pmatrix} \lambda_2 & 0 & \cdots & 0 \\ 0 & \lambda_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{k+1} \end{pmatrix}$$

- (Euclidean) $\mathbf{L}$-embedding of a graph:

$$\mathbf{X} = \mathbf{\Lambda}_k^{-\frac{1}{2}} \mathbf{U}^\top = [\boldsymbol{x}_1 \ \ldots \ \boldsymbol{x}_j \ \ldots \ \boldsymbol{x}_n]$$

The coordinates of a vertex v_j are:

$$\boldsymbol{x}_j = \begin{pmatrix} \frac{\boldsymbol{u}_2(v_j)}{\sqrt{\lambda_2}} \\ \vdots \\ \frac{\boldsymbol{u}_{k+1}(v_j)}{\sqrt{\lambda_{k+1}}} \end{pmatrix}$$

Normalized Laplacian

Why the Scaling?

Both

- the *commute-time distance* (CTD) and
- the *principal-component analysis* of a graph (graph PCA)

are two important concepts; They allow to reason "statistically" on a graph. They are both associated with the *normalized* Laplacian matrix.

Commute-Time Distance (CTD)

- The CTD is a well known quantity in Markov chains;
- It is the average number of (weighted) edges that it takes, starting at vertex v_i, to randomly reach vertex v_j for the first time and go back;
- The CTD decreases as the number of connections between the two nodes increases;
- It captures the connectivity structure of a small graph volume rather than a single path between the two vertices – such as the shortest-path geodesic distance.
- The CTD can be computed in closed form:

$$\mathrm{CTD}^2(v_i, v_j) = \mathrm{vol}(\mathcal{G}) \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$$

vol = sum of degrees

Graph PCA

- The mean (remember that $\sum_{j=1}^{n} \boldsymbol{u}_i(v_j) = 0$):

$$\overline{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_j = \boldsymbol{\Lambda}_k^{-\frac{1}{2}} \begin{pmatrix} \sum_{j=1}^{n} \boldsymbol{u}_2(v_j) \\ \vdots \\ \sum_{j=1}^{n} \boldsymbol{u}_{k+1}(v_j) \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

- The covariance matrix:

$$\mathbf{S} = \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{x}_j \boldsymbol{x}_j^\top = \frac{1}{n}\mathbf{X}\mathbf{X}^\top = \frac{1}{n}\boldsymbol{\Lambda}_k^{-\frac{1}{2}}\mathbf{U}^\top\mathbf{U}\boldsymbol{\Lambda}_k^{-\frac{1}{2}} = \frac{1}{n}\boldsymbol{\Lambda}_k^{-1}$$

- The vectors $\boldsymbol{u}_2, \ldots, \boldsymbol{u}_{k+1}$ are the directions of *maximum variance* of the graph embedding, with $\lambda_2^{-1} \geq \ldots \geq \lambda_{k+1}^{-1}$.

More Laplacian Variants

- The normalized graph Laplacian (symmetric and semi-definite positive):

$$\mathbf{L}_n = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$$

- The transition matrix (allows an analogy with Markov chains):

$$\mathbf{L}_t = \mathbf{D}^{-1}\mathbf{A}$$

- The random-walk graph Laplacian:

$$\mathbf{L}_r = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{L}_t$$

- These matrices are similar:

$$\mathbf{L}_r = \mathbf{D}^{-\frac{1}{2}}\mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}\mathbf{D}^{\frac{1}{2}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}_n\mathbf{D}^{\frac{1}{2}}$$

Eigenvectors/Eigenvalues for L_n, L_r

- $\mathbf{L}_r \boldsymbol{w} = \lambda \boldsymbol{w} \iff \mathbf{L}\boldsymbol{w} = \lambda \mathbf{D}\boldsymbol{w}$, hence:

$$\mathbf{L}_r : \quad \lambda_1 = 0; \quad \boldsymbol{w}_1 = \mathbf{1}$$

- $\mathbf{L}_n \boldsymbol{v} = \lambda \boldsymbol{v}$. By virtue of the similarity transformation between the two matrices:

$$\mathbf{L}_n : \quad \lambda_1 = 0 \quad \boldsymbol{v}_1 = \mathbf{D}^{\frac{1}{2}}\mathbf{1}$$

- More generally, the two matrices have the same eigenvalues:

$$0 = \lambda_1 \leq \ldots \leq \lambda_i \ldots \leq \lambda_n$$

- Their eigenvectors are related by:

$$\boldsymbol{v}_i = \mathbf{D}^{\frac{1}{2}}\boldsymbol{w}_i, \ \forall i = 1 \ldots n$$

Random Walk Spectral Embedding

- The $n \times k$ matrix contains the first k eigenvectors of $\mathbf{L}_r$:

$$\mathbf{W} = \begin{bmatrix} \boldsymbol{w}_2 & \dots & \boldsymbol{w}_{k+1} \end{bmatrix}$$

- It is straightforward to obtain the following expressions, where $\boldsymbol{d}$ and $\mathbf{D}$ are the degree-vector and the degree-matrix:

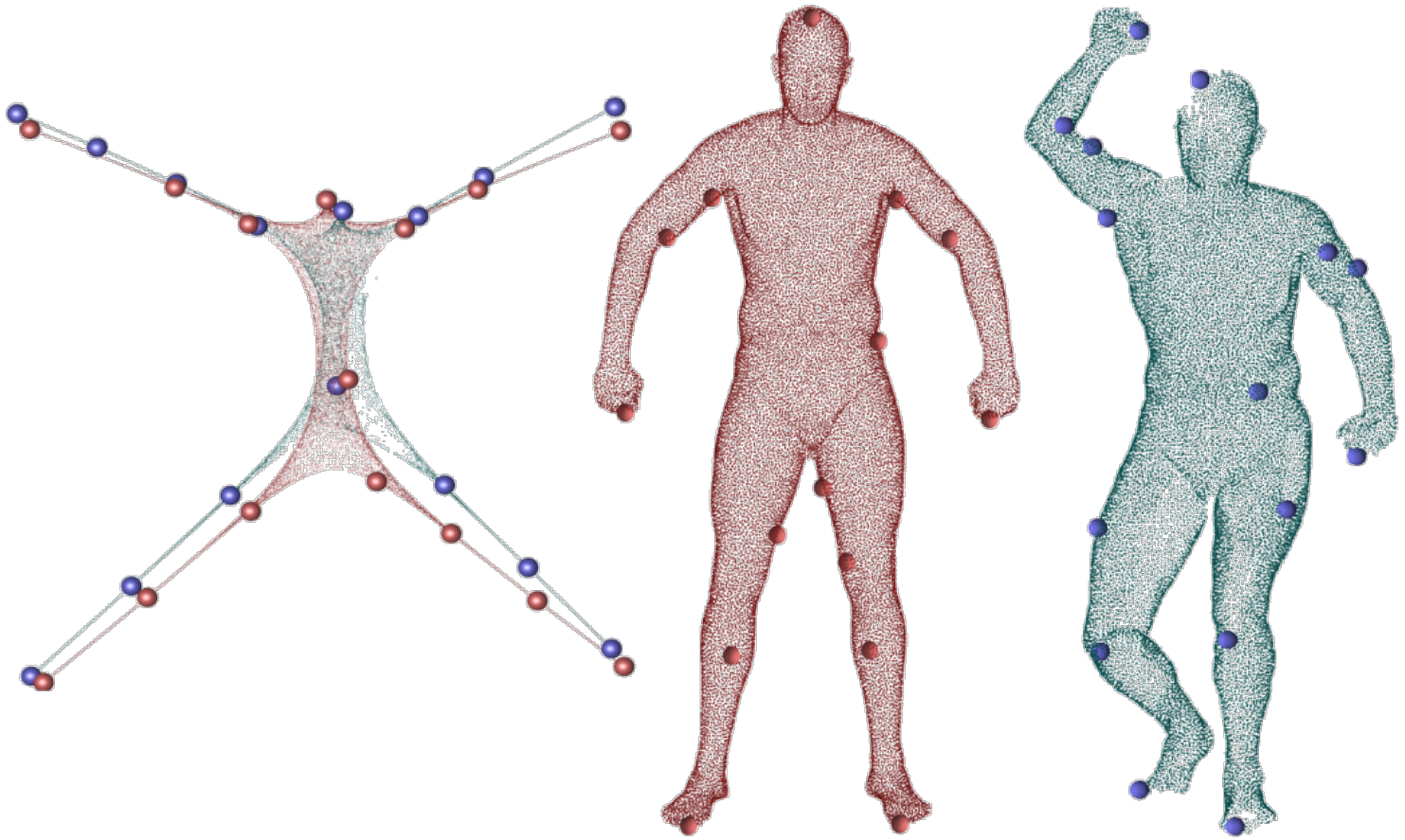
$$\boldsymbol{w}_i^\top \boldsymbol{d} = 0, \ \forall i, 2 \leq i \leq n$$

$$\mathbf{W}^\top \mathbf{D} \mathbf{W} = \mathbf{I}_k$$

- The isometric embedding using the random-walk Laplacian:

$$\mathbf{Y} = \mathbf{W}^\top = \begin{bmatrix} \boldsymbol{y}_1 & \dots & \boldsymbol{y}_n \end{bmatrix}$$

Useful for Isometric Shape Comparisons



Uses random-walk Laplacian

Graph Partitioning

- **The graph-cut problem**: Partition the graph such that:
  1. Edges between groups have very low weight, and
  2. Edges within a group have high weight.

$$\text{cut}(A_1, \ldots, A_k) := \frac{1}{2} \sum_{i=1}^{k} W(A_i, \overline{A}_i) \text{ with } W(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

- **Ratio cut**: (Hagen & Kahng 1992)

$$\text{RatioCut}(A_1, \ldots, A_k) := \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \overline{A}_i)}{|A_i|}$$

- **Normalized cut**: (Shi & Malik 2000)

$$\text{NCut}(A_1, \ldots, A_k) := \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \overline{A}_i)}{\text{vol}(A_i)}$$

vol = sum of degrees

Spectral Clustering

- Both ratio-cut and normalized-cut minimizations are NP-hard problems
- Spectral clustering is a way to solve relaxed versions of these problems:
  1. The smallest non-null eigenvectors of the *unnormalized Laplacian* approximate the RatioCut minimization criterion, and
  2. The smallest non-null eigenvectors of the *random-walk Laplacian* approximate the NCut criterion.

Spectral Clustering Using the Random-Walk Laplacian

- For details see (von Luxburg '07)
- Input: Laplacian $\mathbf{L}_r$ and the number k of clusters to compute.
- Output: Cluster $C_1, \ldots, C_k$.

1. Compute $\mathbf{W}$ formed with the first k eigenvectors of the random-walk Laplacian.
2. Determine the spectral embedding $\mathbf{Y} = \mathbf{W}^\top$
3. Cluster the columns $\boldsymbol{y}_j, j = 1, \ldots, n$ into k clusters using the K-means algorithm.

k-Means Clustering

See Bishop'2006 (pages 424–428) for more details.

- What is a cluster: a group of points whose inter-point distance are small compared to distances to points outside the cluster.
- Cluster centers: $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$.
- Goal: find an assignment of points to clusters as well as a set of vectors $\boldsymbol{\mu}_i$.
- Notations: For each point $\boldsymbol{y}_j$ there is a *binary indicator variable* $r_{ji} \in \{0, 1\}$.
- Objective: minimize the following *distorsion measure*:

$$J = \sum_{j=1}^{n} \sum_{i=1}^{k} r_{ji} \|\boldsymbol{y}_j - \boldsymbol{\mu}_i\|^2$$

k-Means Algorithm

1. Initialization: Choose initial values for $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$.
2. First step: Assign the j-th point to the closest cluster center:

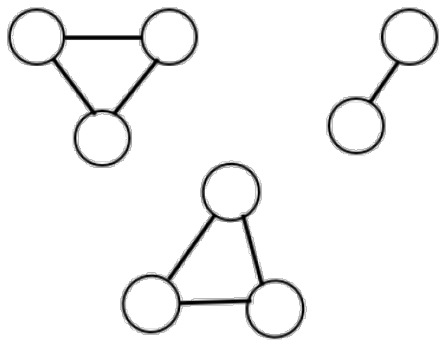
$$r_{ji} = \begin{cases} 1 & \text{if } i = \arg\min_l \|\boldsymbol{y}_j - \mu_l\|^2 \\ 0 & \text{otherwise} \end{cases}$$

3. Second Step: Minimize J to estimate the cluster centers:

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^{n} r_{ji} \boldsymbol{y}_j}{\sum_{j=1}^{n} r_{ji}}$$

4. Convergence: Repeat until no more change in the assignments.

Spectral Clustering: The Ideal Case

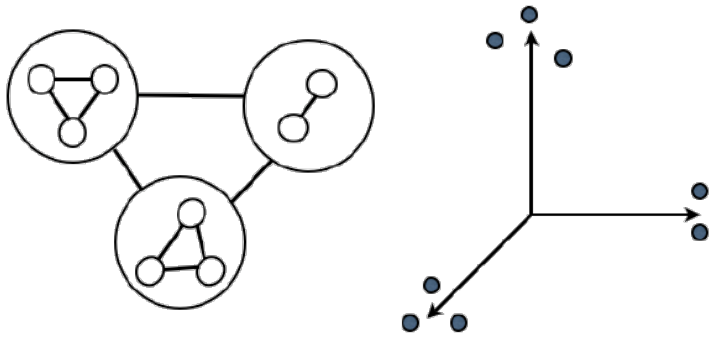


- $\lambda_1 = \lambda_2 = \lambda_3 = 0$
- $\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3$ form an orthonormal basis.
- The connected components collapse to $(100), (010), (001)$.
- Clustering is trivial in this case.

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Spectral Clustering: The Perturbed Case



- See (von Luxburg '07) for a detailed analysis.
- The connected components are no longer *disconnected*, but they are only connected by few edges with low weight.

- The Laplacian is a perturbed version of the ideal case.
- Choosing the first k nonzero eigenvalues is easier the larger the eigengap between $1/\lambda_{k+1}$ and $1/\lambda_{k+2}$.
- The fact that the first k eigenvectors of the perturbed case are approximately piecewise constant depends on $|1/\lambda_{k+1} - 1/\lambda_{k+2}|$.
- Choosing k is a crucial issue.

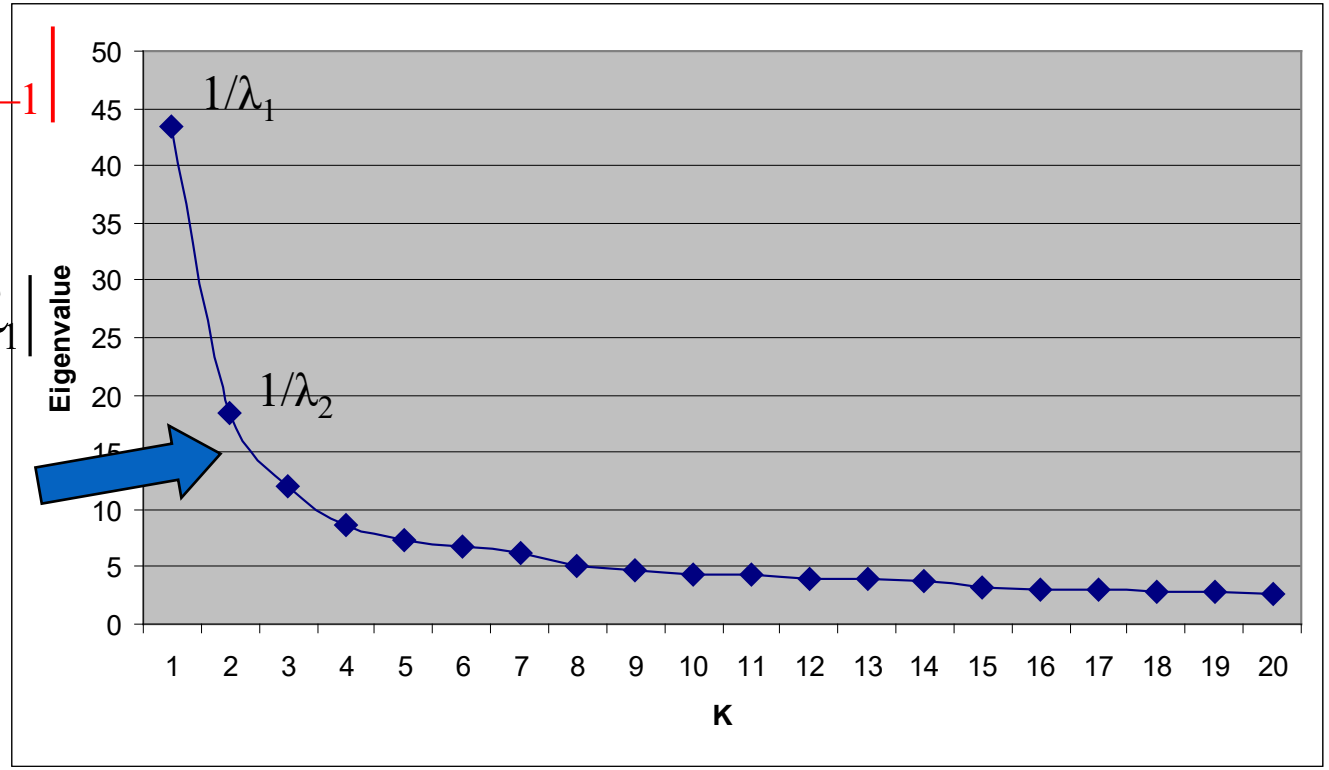
Spectral Gap: Selecting *k*

- *Eigengap*: the difference between two consecutive eigenvalues.
- Most stable clustering is generally given by the value k that maximizes the expression

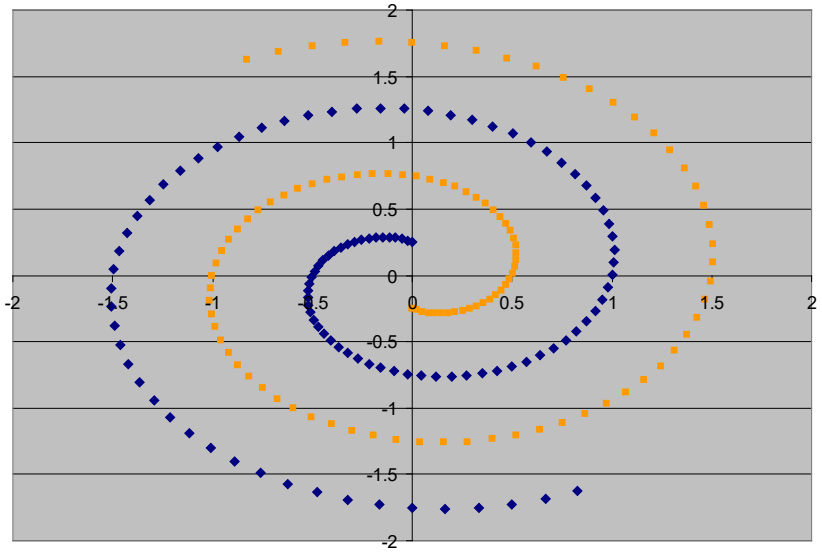
$$\Delta_k = \left| 1/\lambda_k - 1/\lambda_{k-1} \right|$$

$$\max \Delta_k = \left| 1/\lambda_2 - 1/\lambda_1 \right|$$

$\Rightarrow$ Choose $k=2$

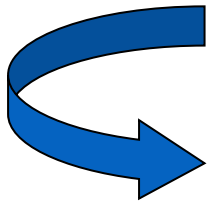


Spirals Again



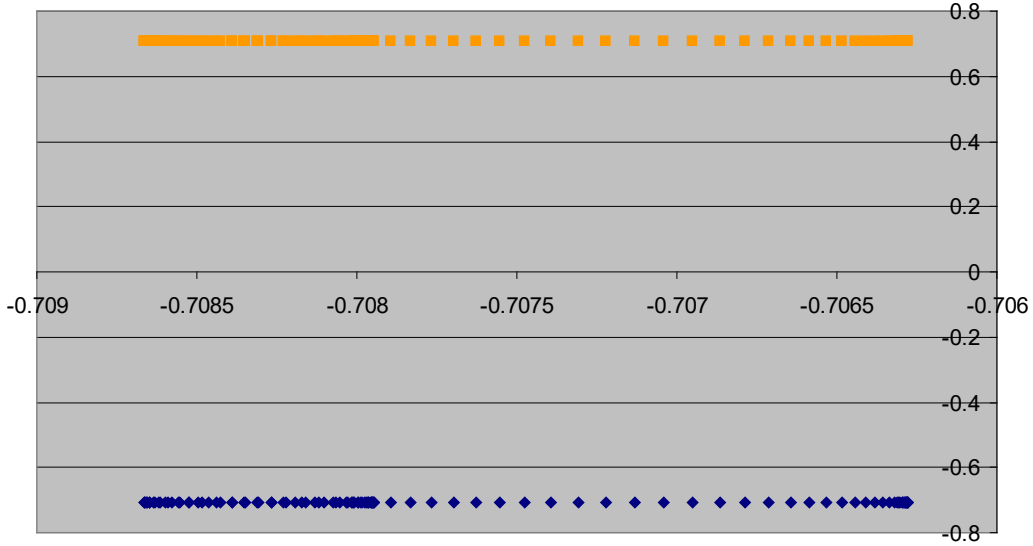
Dataset exhibits complex cluster shapes

$\Rightarrow$ Direct *k*-means performs very poorly in this space due to bias toward dense spherical clusters.



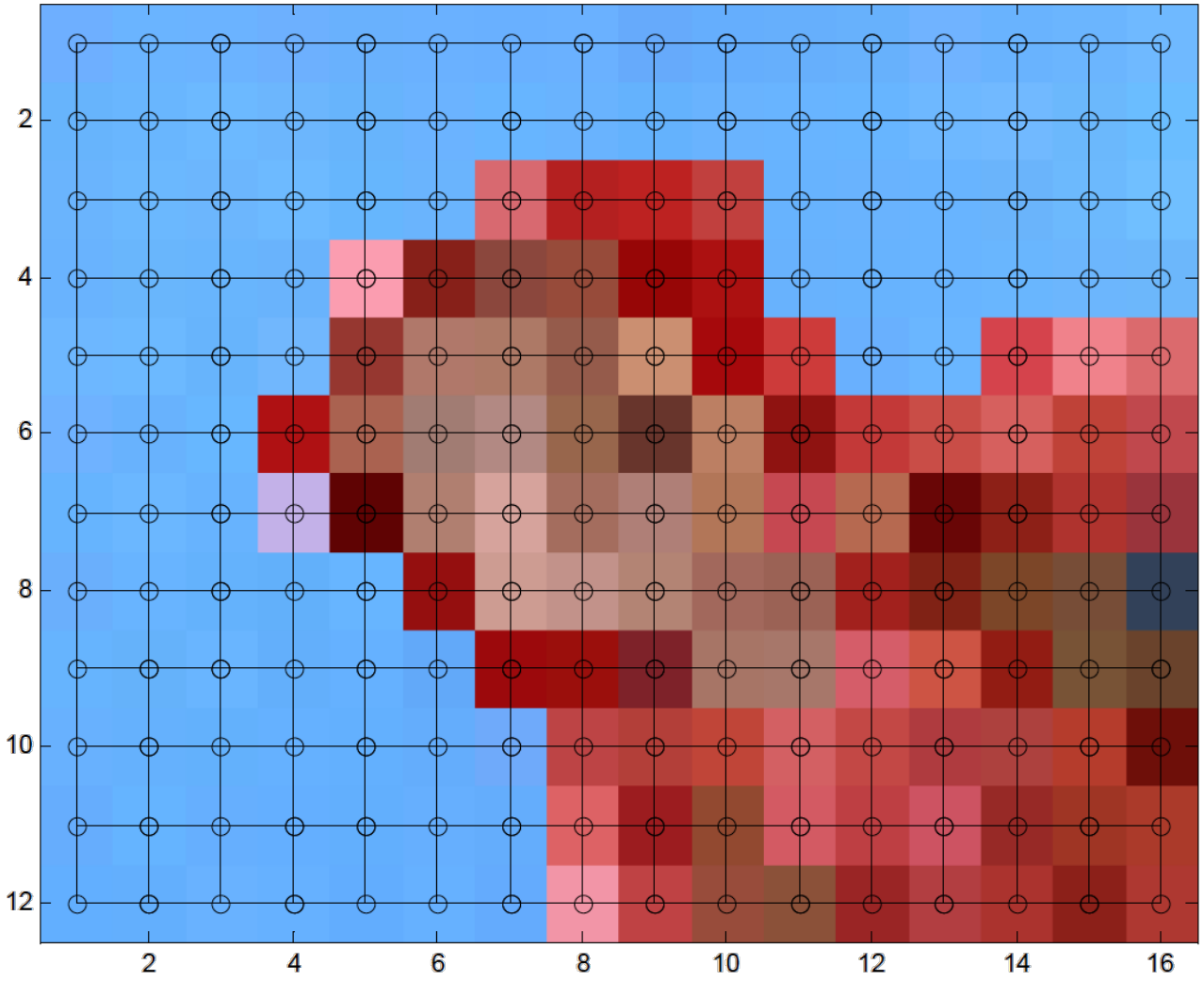
In the embedded space given by two leading eigenvectors, clusters are trivial to separate.

Build nearest neighbor graph

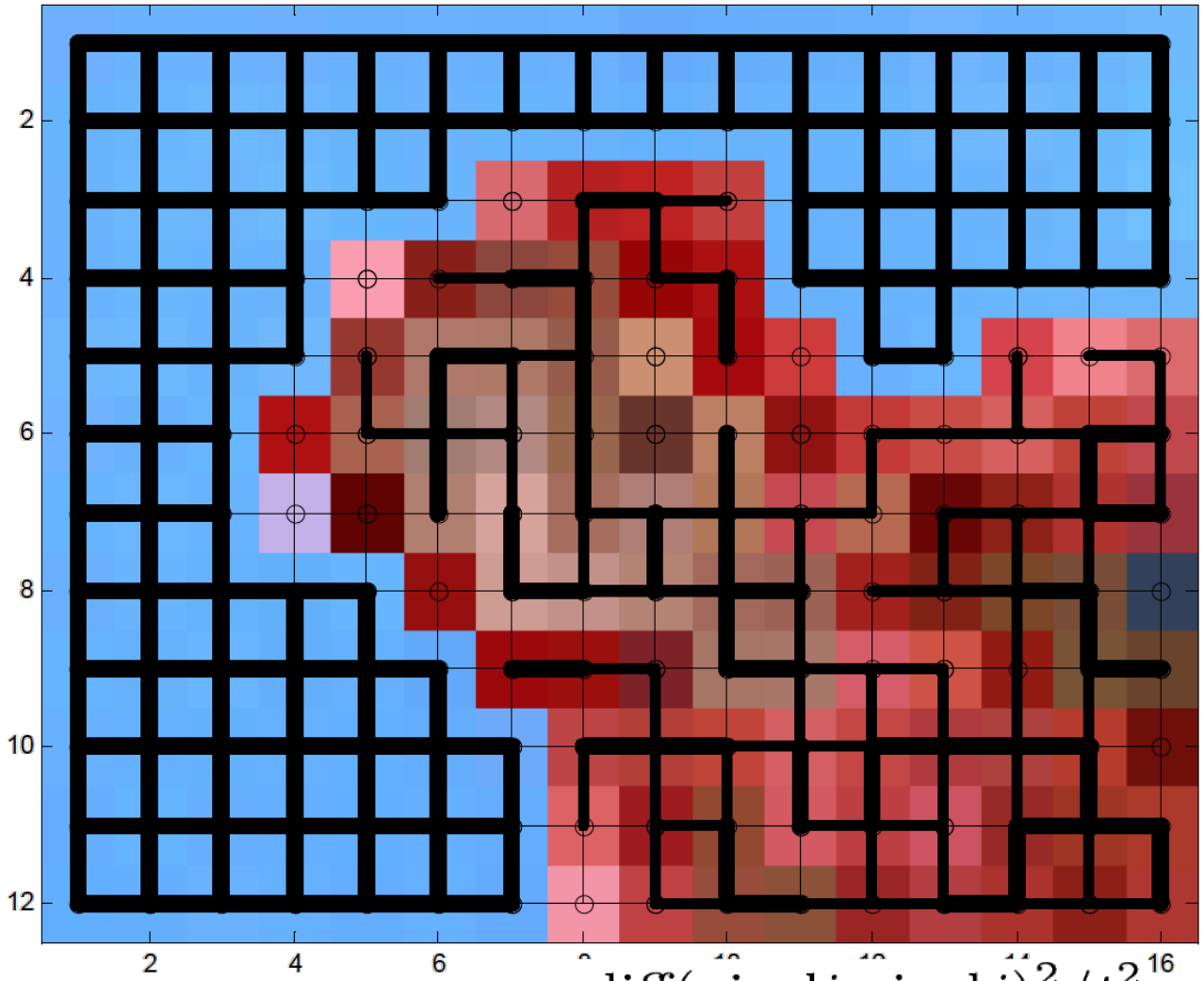


Spectral Image Segmentation (Shi-Malik ’00)

Spectral Image Segmentation (Shi-Malik ’00)

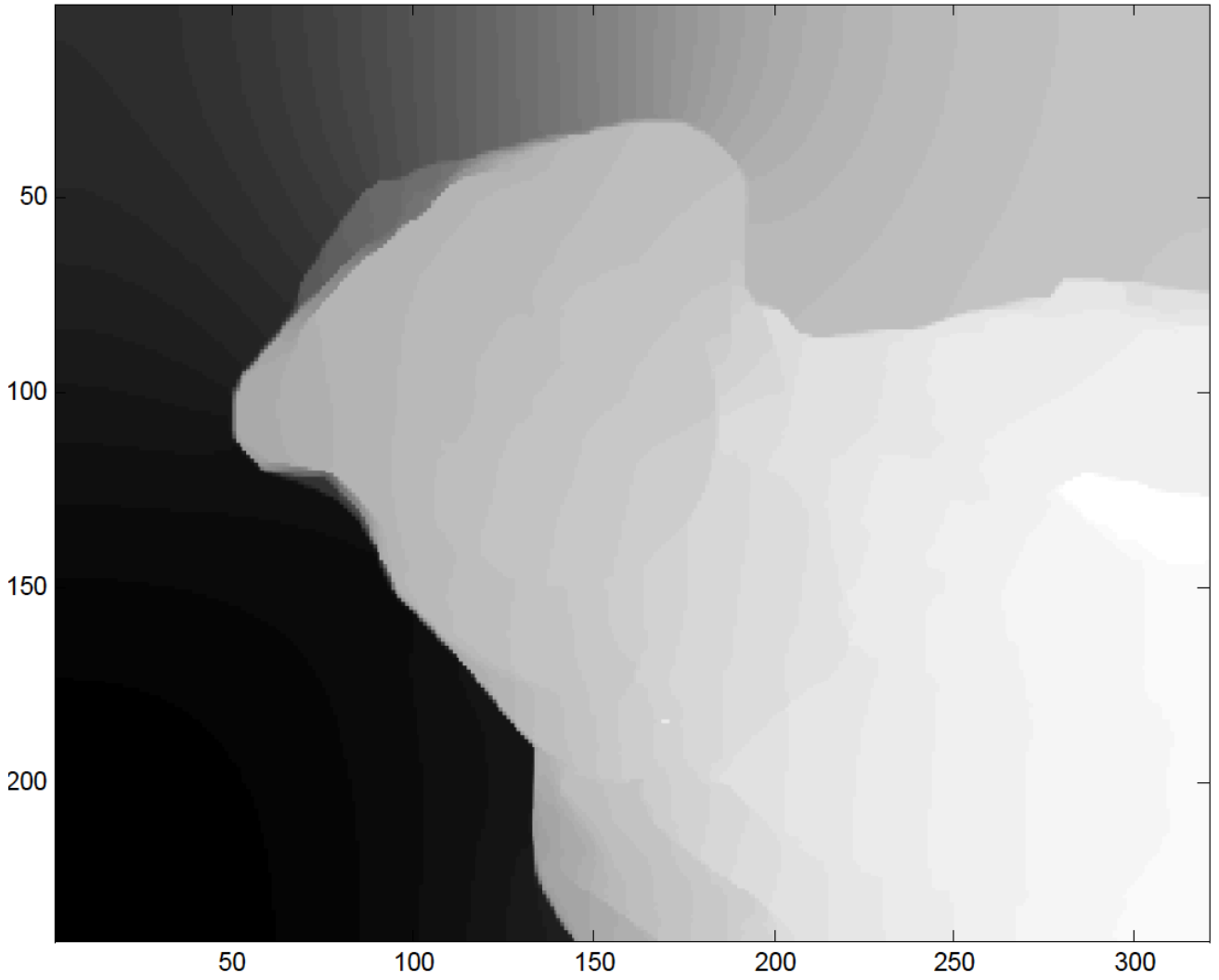


Spectral Image Segmentation (Shi-Malik ’00)



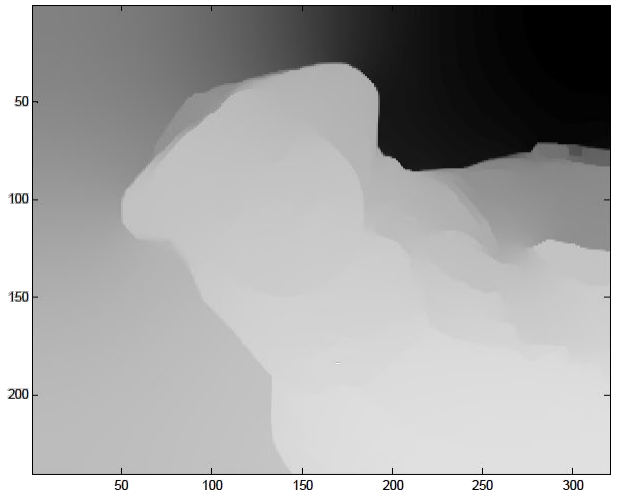
edge weight $e^{-\text{diff}(\text{pixel}i,\text{pixel}j)^2/t^2}$

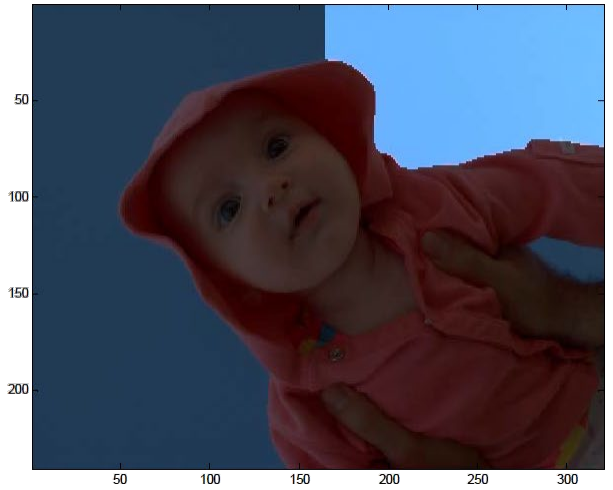
Second Eigenvector

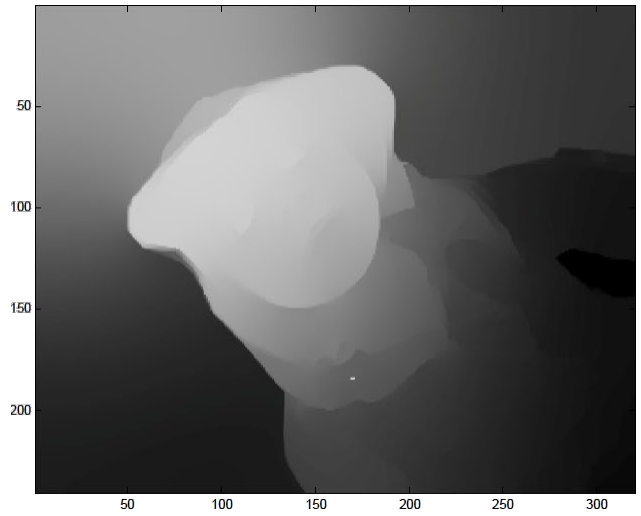


Second Eigenvector Sparsest Cut

3rd and 4th Eigenvectors





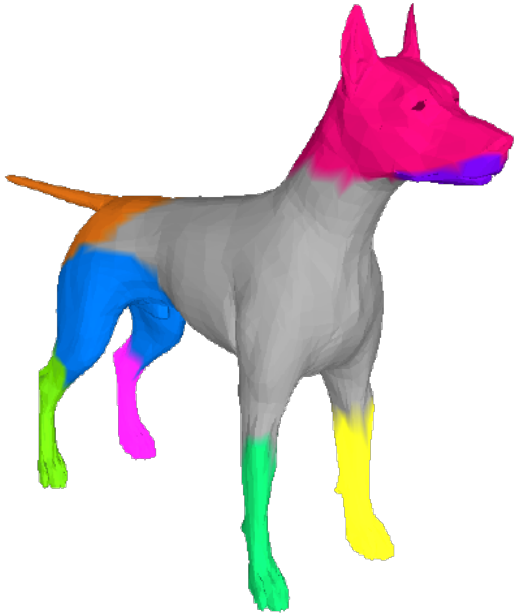


Normalized Cuts

Mesh Segmentation Using Spectral Clustering

K=6

K=6



K=9

K=6

Conclusion

- Spectral graph embedding based on the graph Laplacian is a very powerful tool;
- Allows links between graphs and Riemannian manifolds
- There are strong links with Markov chains and random walks
- It allows clustering (or segmentation) under some conditions

That’s All