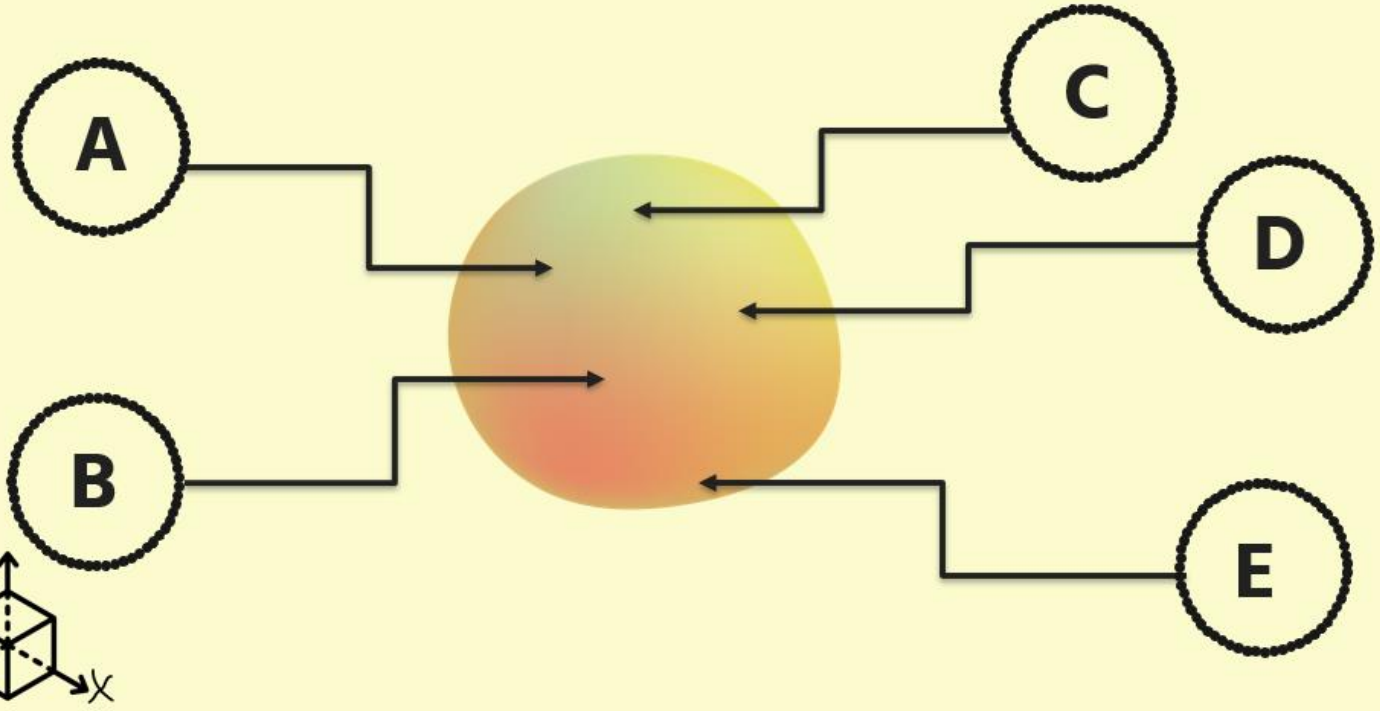
Laplacian Eigenmaps for Dimensionality Reduction

Catherine Barbeau

Graph Construction

Input: A set of n data points

$$\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$$



Graph Construction

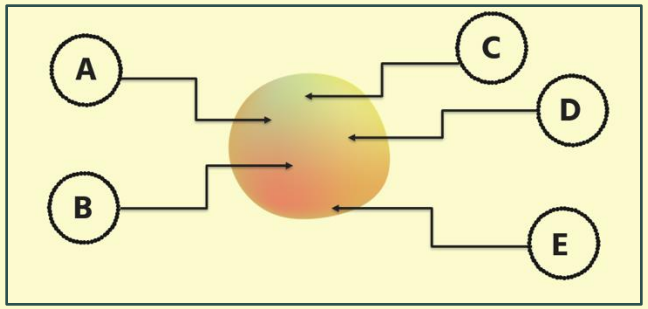
A weighted undirected graph $G = (V, E, W)$

- $V = \{1, 2, \ldots, n\}$ — one node per data point
- E — edges defined via a neighborhood rule (e.g., k-NN or ε-radius)
- $W \in \mathbb{R}^{n \times n}$ — symmetric weight matrix, where:

$$W_{ij} = \begin{cases} \exp\left(-\dfrac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{t}\right), & \text{if } (i,j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

Coordinates of Sampled Points from a 3D Object

Node	x	y	z
A	-1.5	2	4
B	-1	1	2
C	0	0	5
D	2	2	3
E	1	0.5	1



k Nearest Neighbour Graph (kNN Graph)

Nearest Neighbors (Formal Definition):
Given data points $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$ and a distance function $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{\geq 0}$,
the k-nearest neighbors of a point $\mathbf{x}_i$ are defined as:

$$\mathrm{NN}_k(\mathbf{x}_i) = \underset{\substack{S \subseteq \mathcal{X} \setminus \{\mathbf{x}_i\} \\ |S| = k}}{\arg\min} \sum_{\mathbf{x}_j \in S} d(\mathbf{x}_i, \mathbf{x}_j)$$

Note: We use the Euclidean distance in $\mathbb{R}^d$:

Unweighted k-NN Graph Construction:
Construct a graph $G = (V, E)$ where:

- $V = \{1, 2, \ldots, n\}$ corresponds to the data points.
- $(i, j) \in E$ if $\mathbf{x}_j \in \mathrm{NN}_k(\mathbf{x}_i)$ or $\mathbf{x}_i \in \mathrm{NN}_k(\mathbf{x}_j)$.

This yields a symmetric, unweighted graph based on proximity in $\mathbb{R}^d$.

Pairwise Euclidean Distances

$$\mathbb{R}^3 \rightarrow \|x_i - x_j\| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

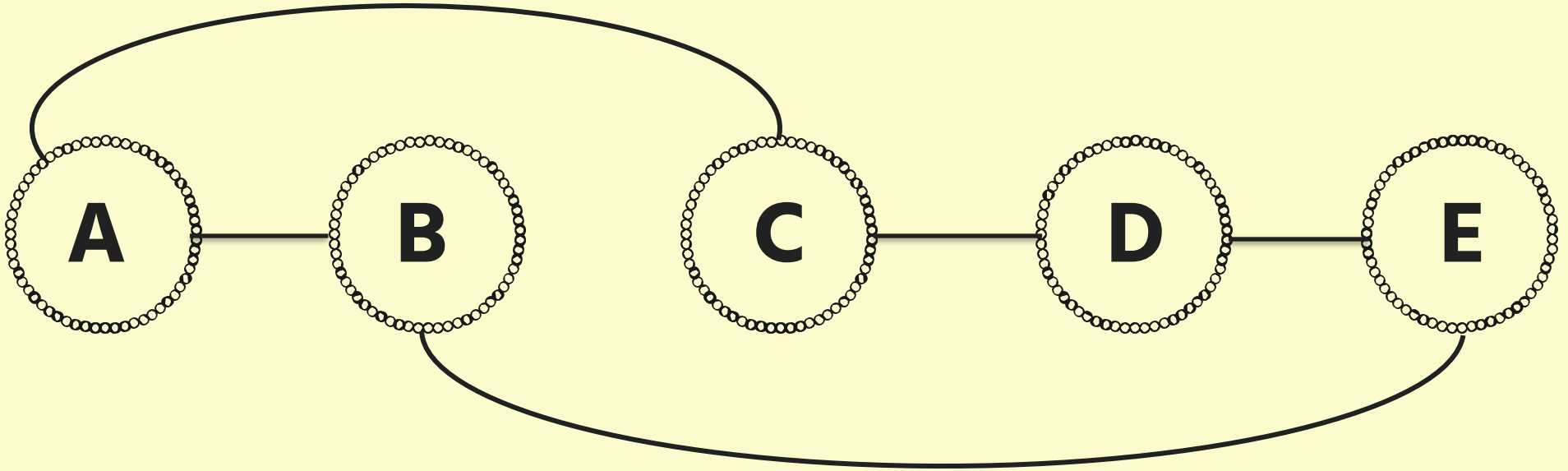
	A	B	C	D	E
A	0.0	2.4	1.8	3.7	3.7
B	2.4	0.0	3.3	3.6	2.4
C	1.8	3.3	0.0	2.8	4.2
D	3.7	3.6	2.8	0.0	2.7
E	3.7	2.4	4.2	2.7	0.0

Pairwise Euclidean Distance Matrix (Heatmap)

	A	B	C	D	E
A	0.0	2.4	1.8	3.7	3.7
B	2.4	0.0	3.3	3.6	2.4
C	1.8	3.3	0.0	2.8	4.2
D	3.7	3.6	2.8	0.0	2.7
E	3.7	2.4	4.2	2.7	0.0

Unweighted kNN Graph

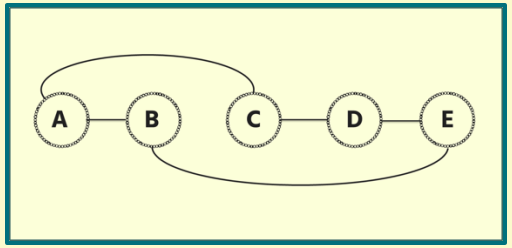
K = 2



Adjacency Matrix Representation

K = 2

	A	B	C	D	E
A	0	**1**	**1**	0	0
B	**1**	0	0	0	**1**
C	**1**	0	0	**1**	0
D	0	0	**1**	0	**1**
E	0	**1**	0	**1**	0



Graph Laplacian

Graph Laplacian (Formal Definition):
Given a graph $G = (V, E)$ with:

- Weight matrix $W \in \mathbb{R}^{n \times n}$, where $W_{ij} > 0$ if $(i, j) \in E$, else 0,
- Degree matrix $D \in \mathbb{R}^{n \times n}$, a diagonal matrix with entries $D_{ii} = \sum_j W_{ij}$,

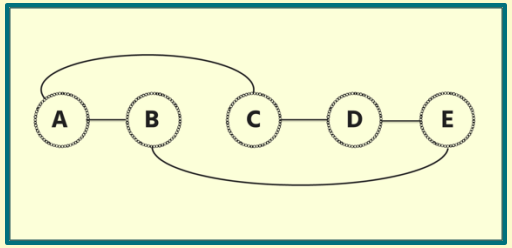
the **(unnormalized) Graph Laplacian** is defined as:

$$L = D - W$$

This matrix captures the local geometry of the data.

Degree Matrix

	A	B	C	D	E
A	**2**	0	0	0	0
B	0	**2**	0	0	0
C	0	0	**2**	0	0
D	0	0	0	**2**	0
E	0	0	0	0	**2**



Binary Weighted Laplacian

	A	B	C	D	E
A	**2**	-1	-1	0	0
B	-1	**2**	0	0	-1
C	-1	0	**2**	-1	0
D	0	0	-1	**2**	-1
E	0	-1	0	-1	**2**

	A	B	C	D	E
A	**2**	0	0	0	0
B	0	**2**	0	0	0
C	0	0	**2**	0	0
D	0	0	0	**2**	0
E	0	0	0	0	**2**

−

	A	B	C	D	E
A	0	**1**	**1**	0	0
B	**1**	0	0	0	**1**
C	**1**	0	0	**1**	0
D	0	0	**1**	0	**1**
E	0	**1**	0	**1**	0

Heat Kernel

Heat Kernel Weighting:
The **heat kernel** is a weighting mechanism which assigns higher weights to nearby points and lower weights to distant ones.

Formally, for $(i, j) \in E$, the edge weight is defined as:

$$W_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{t}\right)$$

where $t > 0$ is a scaling parameter.

Heat Kernel Weight Matrix Construction:
For data points $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, we define the weight matrix $W \in \mathbb{R}^{n \times n}$ as:

$$W_{ij} = \begin{cases} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{t}\right), & \text{if } (i, j) \in E \\ 0, & \text{otherwise} \end{cases}$$

where E is the set of edges (based on k-nearest neighbors).

Heat Kernel Weight Matrix

	A	B	C	D	E
A	0	**0.0031**	**0.0392**	0	0
B	**0.0031**	0	0	0	**0.0031**
C	**0.0392**	0	0	**0.00039**	0
D	0	0	**0.00039**	0	**0.00068**
E	0	**0.0031**	0	**0.00068**	0

$$W_{ij} = \begin{cases} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{t}\right), & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$

Heat Kernel Degree Matrix

	A	B	C	D	E
A	**0.0423**	0	0	0	0
B	0	**0.0062**	0	0	0
C	0	0	**0.03959**	0	0
D	0	0	0	**0.00107**	0
E	0	0	0	0	**0.00378**

Heat Kernel Weighted Graph Laplacian

	A	B	C	D	E
A	**0.0423**	**-0.0031**	**-0.0392**	0	0
B	**-0.0031**	**0.0062**	0	0	**-0.0031**
C	**-0.0392**	0	**0.03959**	**-0.00039**	0
D	0	0	**-0.00039**	**0.00107**	**-0.00068**
E	0	**-0.0031**	0	**-0.00068**	**0.00378**

	A	B	C	D	E
A	**0.0423**	0	0	0	0
B	0	**0.0062**	0	0	0
C	0	0	**0.03959**	0	0
D	0	0	0	**0.00107**	0
E	0	0	0	0	**0.00378**

−

	A	B	C	D	E
A	0	**0.0031**	**0.0392**	0	0
B	**0.0031**	0	0	0	**0.0031**
C	**0.0392**	0	0	**0.00039**	0
D	0	0	**0.00039**	0	**0.00068**
E	0	**0.0031**	0	**0.00068**	0

Generalized Eigenvalue Problem

Generalized Eigenvalue Problem:

$$L\mathbf{y} = \lambda D\mathbf{y}$$

Where:

- $L = D - W$: Graph Laplacian
- $D_{ii} = \sum_j W_{ij}$: Degree matrix (diagonal)
- $\lambda \in \mathbb{R}$: Eigenvalue — quantifies the variation of $\mathbf{y}$ over the graph
- $\mathbf{y} \in \mathbb{R}^n$: Eigenvector — a non-zero solution that satisfies the equation above

Solving the Generalized Eigenvalue Problem

$$Lx = \lambda Dx$$

$$\begin{bmatrix} 0.0423 & -0.0031 & -0.0392 & 0 & 0 \\ -0.0031 & 0.0062 & 0 & 0 & -0.0031 \\ -0.0392 & 0 & 0.03959 & -0.00039 & 0 \\ 0 & 0 & -0.00039 & 0.00107 & -0.00068 \\ 0 & -0.0031 & 0 & -0.00068 & 0.00378 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \lambda \begin{bmatrix} 0.0423 & 0 & 0 & 0 & 0 \\ 0 & 0.0062 & 0 & 0 & 0 \\ 0 & 0 & 0.03959 & 0 & 0 \\ 0 & 0 & 0 & 0.00107 & 0 \\ 0 & 0 & 0 & 0 & 0.00378 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

Solve using an online equation solver or python script.
Attached is the code I used to solve this example:

Eigenvalue Example Solver Script

Solutions From the Script

For $d = 2$, we use the first two non-trivial eigenvectors:

$$\lambda_1 = 0.3085, \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0.0632 \\ -0.4436 \\ 0.0822 \\ -0.5785 \\ -0.6766 \end{bmatrix}$$

$$\lambda_2 = 0.9902, \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0.0095 \\ 0.2142 \\ -0.0168 \\ -0.9766 \\ -0.0054 \end{bmatrix}$$

Constructing the Low-Dimensional Embedding

Low-Dimensional Embedding:
Given the solved generalized eigenvalue problem:

$$L\mathbf{y} = \lambda D\mathbf{y}$$

Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the ordered eigenvalues with corresponding eigenvectors $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$.

Discard the trivial eigenvector $\mathbf{y}_1 \propto \mathbf{1}$, and construct the embedding using the next d eigenvectors:

$$\Phi(\mathbf{x}_i) = \begin{bmatrix} \mathbf{y}_2(i) \\ \mathbf{y}_3(i) \\ \vdots \\ \mathbf{y}_{d+1}(i) \end{bmatrix} \in \mathbb{R}^d$$

This maps each node i to a point in $\mathbb{R}^d$, preserving the local structure of the graph.

Constructing the Low-Dimensional Embedding

In our example, we choose $d = 2$. Therefore, the embedding function Φ takes the form:

$$\Phi(\mathbf{x}_i) = \begin{bmatrix} y_2(i) \\ y_3(i) \end{bmatrix} \in \mathbb{R}^2$$

A

$$\begin{bmatrix} 0.0632 \\ 0.0095 \end{bmatrix}$$

B

$$\begin{bmatrix} -0.4436 \\ 0.2142 \end{bmatrix}$$

C

$$\begin{bmatrix} 0.0822 \\ -0.0168 \end{bmatrix}$$

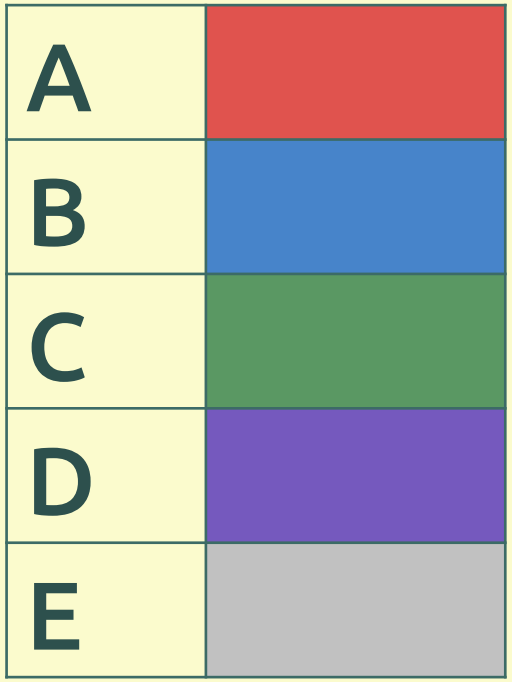
D

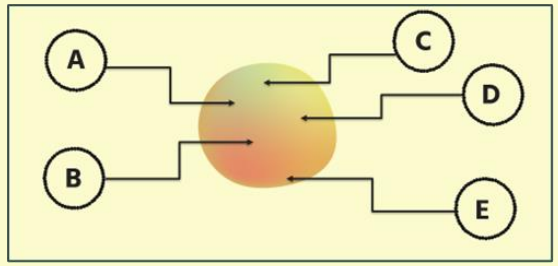
$$\begin{bmatrix} -0.5785 \\ -0.9766 \end{bmatrix}$$

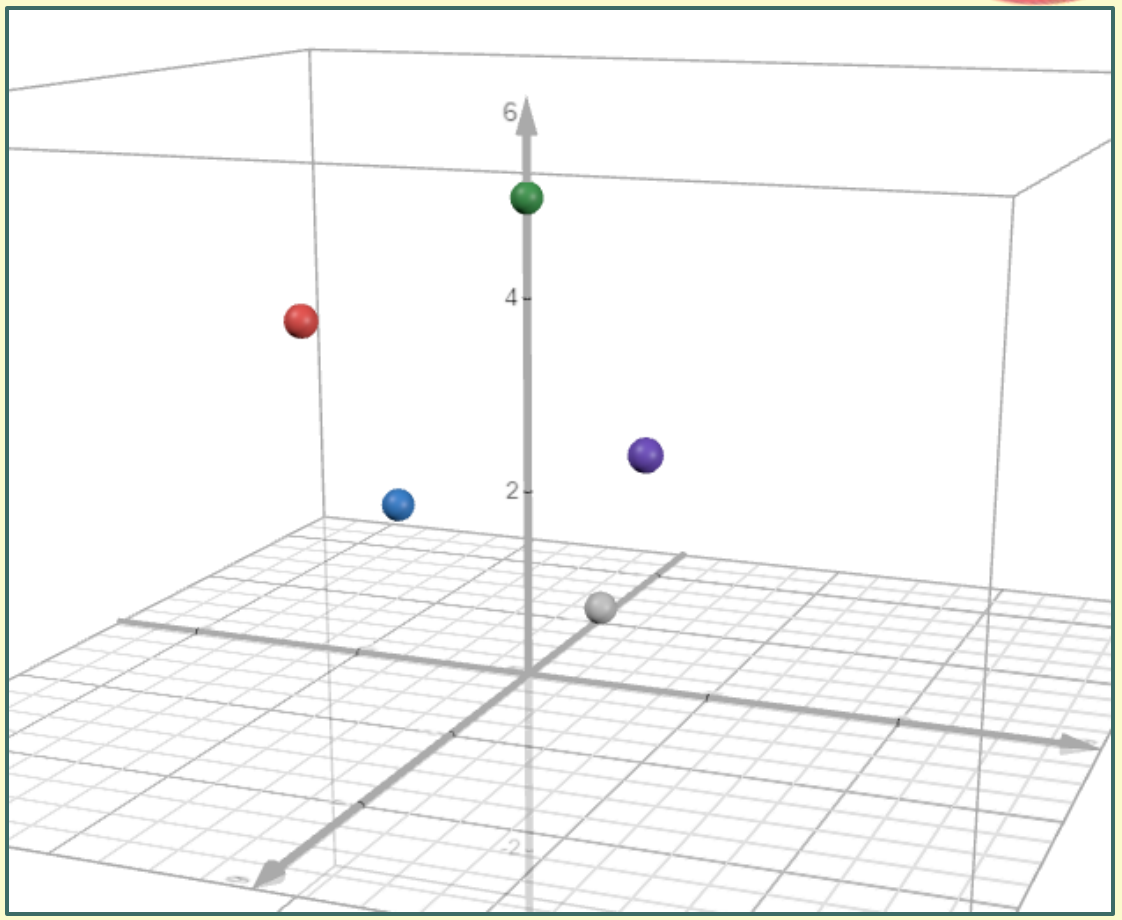
E

$$\begin{bmatrix} -0.6766 \\ -0.0054 \end{bmatrix}$$

Recap of the Original Embeddings

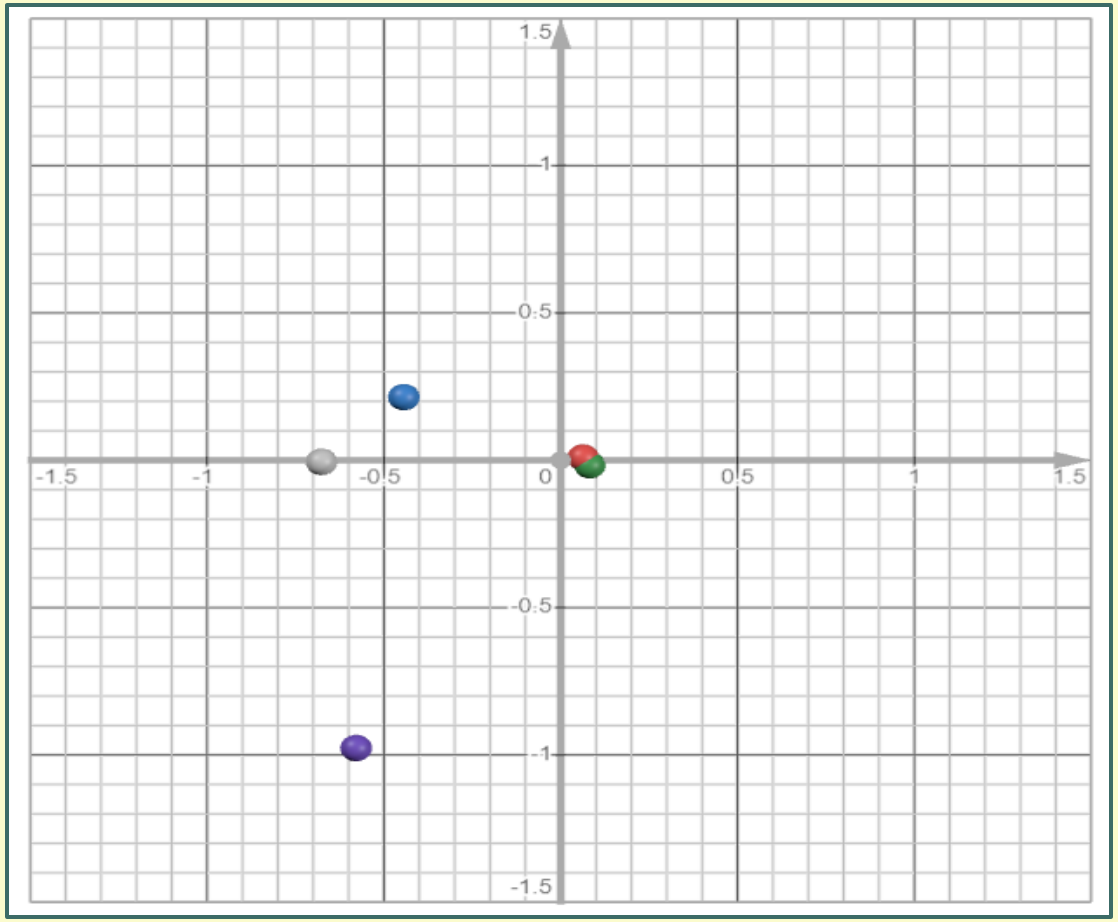






New Lower Dimensional Embeddings

A	
B	
C	
D	
E	



Convergence Guarantee / Analysis

Clarification: What converges?
It's not the algorithm that converges — it's the **graph Laplacian**.
As we sample more data points:

$$L_n \longrightarrow \Delta_{\mathcal{M}}$$

That is, the discrete Laplacian L_n (from our finite graph) approximates the continuous Laplace–Beltrami operator $\Delta_{\mathcal{M}}$ on the manifold.

Why this matters for our algorithm:
As we sample more data points, the discrete graph Laplacian we compute approximates the Laplace–Beltrami operator. This ensures the eigenvectors we use for embedding reflect the true geometry of the underlying manifold.

Theoretical guarantee:
This convergence is formally proven in Belkin and Niyogi's 2005 follow-up paper: *"Towards a theoretical foundation for Laplacian Eigenmaps"*.

Full Algorithm for Reference (1)

Input: Data points $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^D$, embedding dimension d, neighborhood size k, heat kernel parameter t
Output: Low-dimensional embeddings $\Phi(\mathbf{x}_i) \in \mathbb{R}^d$ for each point

Step 1: Construct the kNN graph
Connect each data point $\mathbf{x}_i$ to its k nearest neighbors to form an undirected graph $G = (V, E)$.

Step 2: Compute the graph Laplacian
Assign edge weights using the heat kernel:

$$W_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{t}\right) \quad \text{if } \mathbf{x}_j \in \text{kNN}(\mathbf{x}_i), \text{ else } 0.$$

Compute the degree matrix $D_{ii} = \sum_j W_{ij}$ and Laplacian $L = D - W$.

Full Algorithm for Reference (2)

Step 3: Extract the d lowest non-trivial eigenvectors
Solve the generalized eigenvalue problem:

$$L\mathbf{y} = \lambda D\mathbf{y},$$

and keep the d eigenvectors corresponding to the smallest non-zero eigenvalues.

Step 4: Use the eigenvectors to embed the manifold
Let $\mathbf{y}_1, \ldots, \mathbf{y}_d$ be the selected eigenvectors. Then the low-dimensional embedding of point $\mathbf{x}_i$ is:

$$\Phi(\mathbf{x}_i) = \begin{bmatrix} y_1(i) \\ \vdots \\ y_d(i) \end{bmatrix} \in \mathbb{R}^d.$$

Bibliography

[1] M. Belkin and P. Niyogi, *Laplacian Eigenmaps for Dimensionality Reduction and Data Representation*, Neural Computation, vol. 15, no. 6, pp. 1373–1396, 2003.

[2] M. Belkin and P. Niyogi, *Towards a Theoretical Foundation for Laplacian Eigenmaps*, in Learning Theory: 18th Annual Conference on Computational Learning Theory (COLT), 2005.

[3] M. M. Deza and E. Deza, *Encyclopedia of Distances*, Springer, 2009. [For k-Nearest Neighbor Graphs]

[4] F. R. K. Chung, *Spectral Graph Theory*, American Mathematical Society, 1997. [For Graph Laplacians]

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins University Press, 2013. [For Generalized Eigenvalue Problems]

[6] S. Rosenberg, *The Laplacian on a Riemannian Manifold*, Cambridge University Press, 1997. [For Laplace-Beltrami Operator]