# Laplacian Eigenmaps for Dimensionality Reduction

## Differential Methods for AI

Sergio Herreros Pérez, Mario Kroll Merino, Carlos Mazuecos Reíllo,
José Andrés Ridruejo Tuñón

# Laplacian Eigenmaps: The Core Intuition

## Goal

Map high-dimensional data $x_1, \ldots, x_N \in \mathbb{R}^D$ to low-dimensional points $\mathbf{y}_1, \ldots, \mathbf{y}_N \in \mathbb{R}^m$ (where $m \ll D$) while preserving **local neighborhood information**.

**The Fundamental Idea:**

- If two points $x_i$ and $x_j$ are close in the original space (high similarity), they should be mapped close together in the new space.
- We treat this as an optimization problem minimizing the stretching of edges in the neighborhood graph.

# Step 1: Graph Construction & Weights

First, we define the geometry of the data manifold.

1. **Adjacency (Neighbors):** Use $k$-Nearest Neighbors (k-NN). To ensure the graph is undirected, we enforce symmetry:

$$\text{Connect } i \text{ and } j \text{ if } i \in kNN(j) \textbf{ OR } j \in kNN(i).$$

2. **Weights ($W$):** Assign weights $W_{ij}$ (Heat Kernel):

$$W_{ij} = e^{-\frac{||x_i - x_j||^2}{t}} \quad \text{if connected, else 0.}$$

3. **Symmetrization:** Ensure $W$ is symmetric ($W_{ij} = W_{ji}$) so eigenvalues are real.
4. **Degree Matrix ($D$):** Diagonal matrix measuring density.

$$D_{ii} = \sum_j W_{ij}$$

# Step 2: The Optimization Problem (1D Case)

Let's try to map the data to a **single line** ($m = 1$). Let $\mathbf{y} = (y_1, y_2, \ldots, y_N)^T$ be the coordinate vector.
We want to minimize the weighted distance between neighbors:

## Objective Function

$$\min_{\mathbf{y}} \sum_{i,j} (y_i - y_j)^2 W_{ij}$$

- If $W_{ij}$ is large (points are close), $y_i$ and $y_j$ **must** be close to minimize the cost.
- If $W_{ij} = 0$, the distance $(y_i - y_j)$ doesn't matter.

# Step 3: From Summation to Matrix Form

We can rewrite the sum using algebraic manipulation:

$$\begin{aligned}\sum_{i,j}(y_i - y_j)^2 W_{ij} &= \sum_{i,j}(y_i^2 + y_j^2 - 2y_i y_j) W_{ij} \\ &= \sum_i y_i^2 D_{ii} + \sum_j y_j^2 D_{jj} - 2\sum_{i,j} y_i y_j W_{ij} \\ &= 2\mathbf{y}^T D\mathbf{y} - 2\mathbf{y}^T W\mathbf{y} \\ &= 2\mathbf{y}^T (D - W)\mathbf{y}\end{aligned}$$

## The Graph Laplacian

Defining $L = D - W$, the objective function becomes:

$$\text{Cost}(\mathbf{y}) = 2\mathbf{y}^T L\mathbf{y}$$

# Step 4: Constraints and Lagrange Multipliers

To prevent the trivial solution ($\mathbf{y} = \mathbf{0}$, cost=0), we impose a constraint to fix the scale and handle density:

$$\mathbf{y}^T D\mathbf{y} = 1$$

Now we solve using **Lagrange Multipliers**:

$$\mathcal{L}(\mathbf{y}, \lambda) = \mathbf{y}^T L\mathbf{y} - \lambda(\mathbf{y}^T D\mathbf{y} - 1)$$

Taking the derivative with respect to $\mathbf{y}$ and setting to 0:

$$2L\mathbf{y} - 2\lambda D\mathbf{y} = 0 \implies \boxed{L\mathbf{y} = \lambda D\mathbf{y}}$$

This is the **Generalized Eigenvalue Problem**.

# Step 5: Why Smallest Eigenvalues?

We have found that the optimal $\mathbf{y}$ must be an eigenvector. But which one? Let's check the cost of a specific eigenvector solution $\mathbf{y}$:

$$\text{Cost} = \mathbf{y}^T L \mathbf{y}$$

Since $L\mathbf{y} = \lambda D\mathbf{y}$, we substitute:

$$\mathbf{y}^T L \mathbf{y} = \mathbf{y}^T (\lambda D \mathbf{y}) = \lambda (\mathbf{y}^T D \mathbf{y})$$

Using our constraint $\mathbf{y}^T D\mathbf{y} = 1$:

$$\text{Cost} = \lambda$$

**Conclusion**

To **minimize the cost**, we must choose the eigenvectors with the **smallest eigenvalues** $\lambda$.

# Step 6: Generalizing to $m$ Dimensions

- $\lambda_0 = 0$: Corresponds to eigenvector $\mathbf{1}$ (constant vector). Maps all points to a single spot. **Discard it.**
- $\lambda_1, \ldots, \lambda_m$: The smallest non-zero eigenvalues.

**The Embedding Map:** For embedding into $m$ dimensions, we use the eigenvectors $\mathbf{f}_1, \ldots, \mathbf{f}_m$ associated with $0 < \lambda_1 \leq \cdots \leq \lambda_m$.

$$x_i \mapsto (\mathbf{f}_1(i), \mathbf{f}_2(i), \ldots, \mathbf{f}_m(i))$$

## Summary

The coordinates in the low-dimensional space are literally the values of the Laplacian eigenvectors.

# From Continuous to Discrete: The Key Connection

**Continuous Laplacian**

$$-\Delta u = \lambda u$$

$$\int |\nabla u|^2$$

**Graph Laplacian**

$$Lf = \lambda f$$

$$f^\top Lf = \sum w_{ij}(f_i - f_j)^2$$

## Conceptual Correspondence

- $|\nabla u|^2 \longleftrightarrow (f_i - f_j)^2$
- Laplace–Beltrami operator $\longleftrightarrow$ Graph Laplacian
- Smoothness on the manifold $\longleftrightarrow$ Smoothness on the graph

# Laplacian Eigenmaps in Graph-Based Learning

- Laplacian Eigenmaps provide the spectral foundation for many graph learning algorithms:
  - **Spectral Clustering**: uses the smallest non-zero Laplacian eigenvectors to partition graphs.
  - **Graph Convolutional Networks (GCNs)**: graph convolutions are defined using the Laplacian spectrum.
- The Laplacian encodes the local geometry of the dataset by capturing similarity relations through weights $w_{ij}$.
- Eigenvectors provide smooth representations that respect this geometry.

# Connection to the Finite Element Method (FEM)

## Shared Variational Principle

Both FEM and Laplacian Eigenmaps are based on minimizing an energy associated with the Laplacian:

$$\int_{\Omega} |\nabla u|^2 \quad \longleftrightarrow \quad \sum_{i,j} w_{ij}(f_i - f_j)^2 = f^\top L f.$$

- FEM approximates $\Delta$ on a mesh; Laplacian Eigenmaps approximate $\Delta$ on a **similarity graph**.
- In both cases, low-energy eigenfunctions correspond to the smoothest modes.
- This provides a rigorous bridge between PDE-based models and graph learning.

# From Continuous Geometry to Discrete Data

- Laplacian Eigenmaps allow transferring ideas from differential geometry to data analysis:
  - **Diffusion**: heat flow on a manifold $\leftrightarrow$ diffusion processes on graphs.
  - **Smoothness**: low-oscillation eigenfunctions $\leftrightarrow$ low-variation graph signals.
  - **Eigenmodes**: Laplace–Beltrami eigenfunctions $\leftrightarrow$ graph Laplacian eigenvectors.
- This creates a unified framework connecting:
  - PDEs and variational principles,
  - manifold geometry,
  - and machine learning on graphs.
- The result: geometric structure of data becomes accessible even in discrete, high-dimensional settings.

# Swiss Roll Dataset

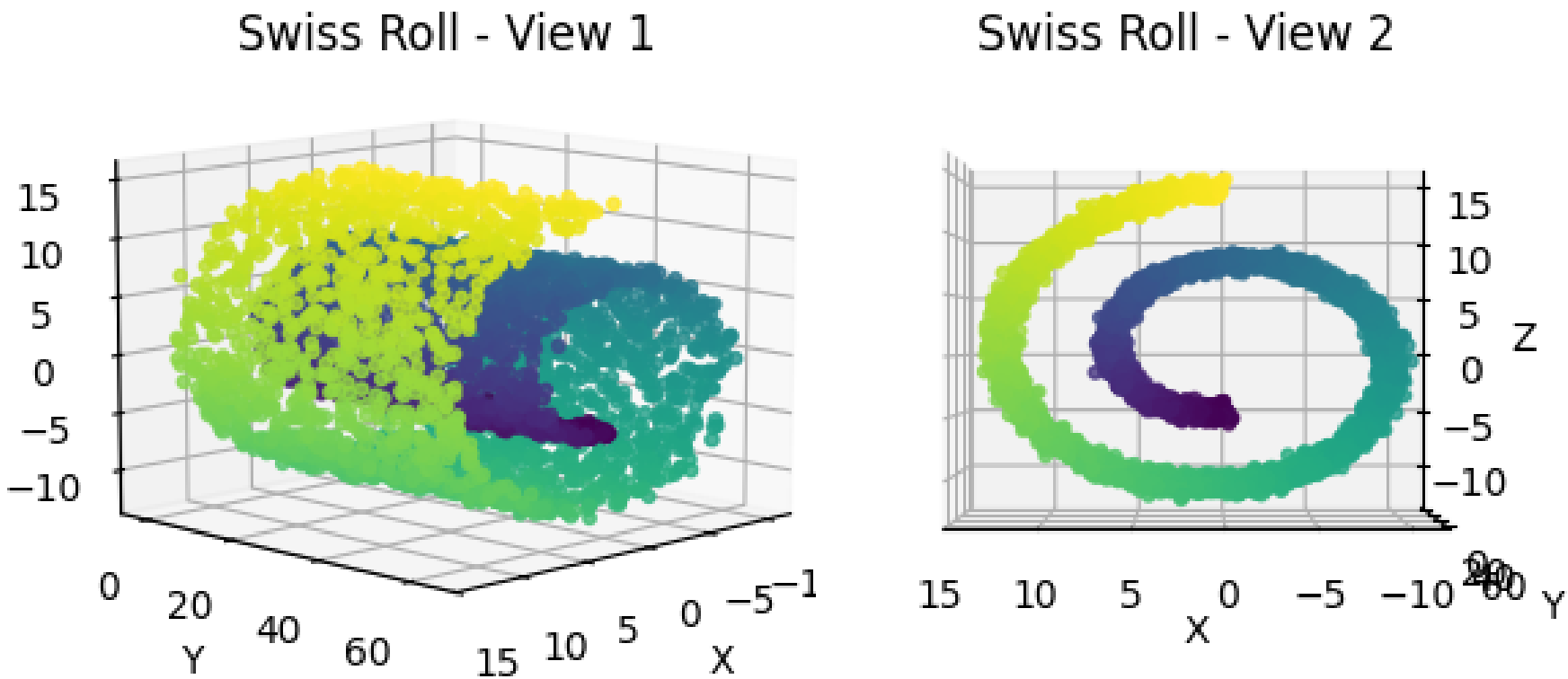


Figure: 3D Visualization of the Swiss Roll Dataset

# Laplacian Eigenmaps Embedding of Swiss Roll

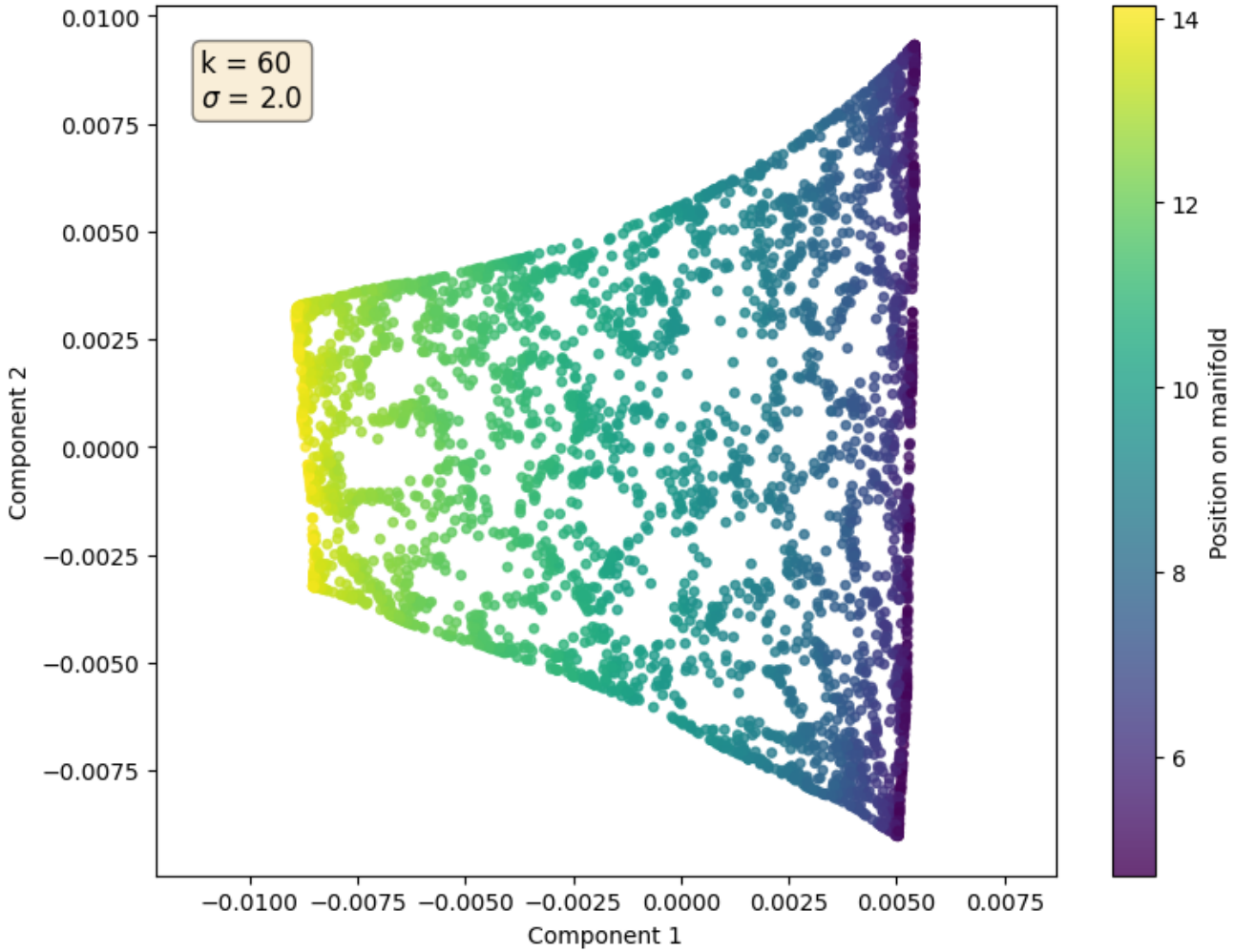


Figure: 2D Embedding of the Swiss Roll Dataset using Laplacian Eigenmaps

# Effect of Neighbors (k) on Embedding

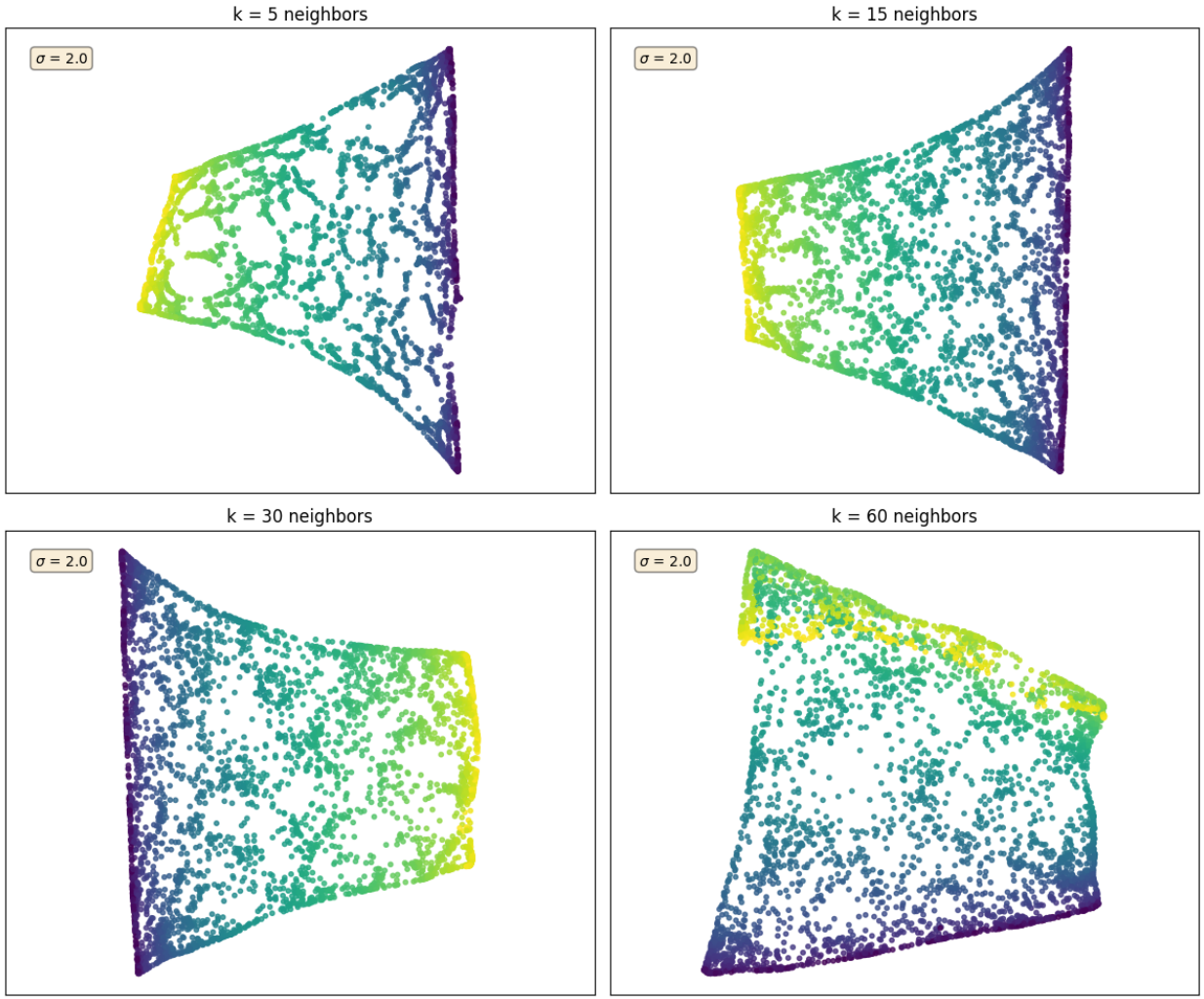


Figure: Effect of Varying Number of Neighbors (k) on the Embedding

# Effect of Sigma (Kernel Bandwidth) on Embedding

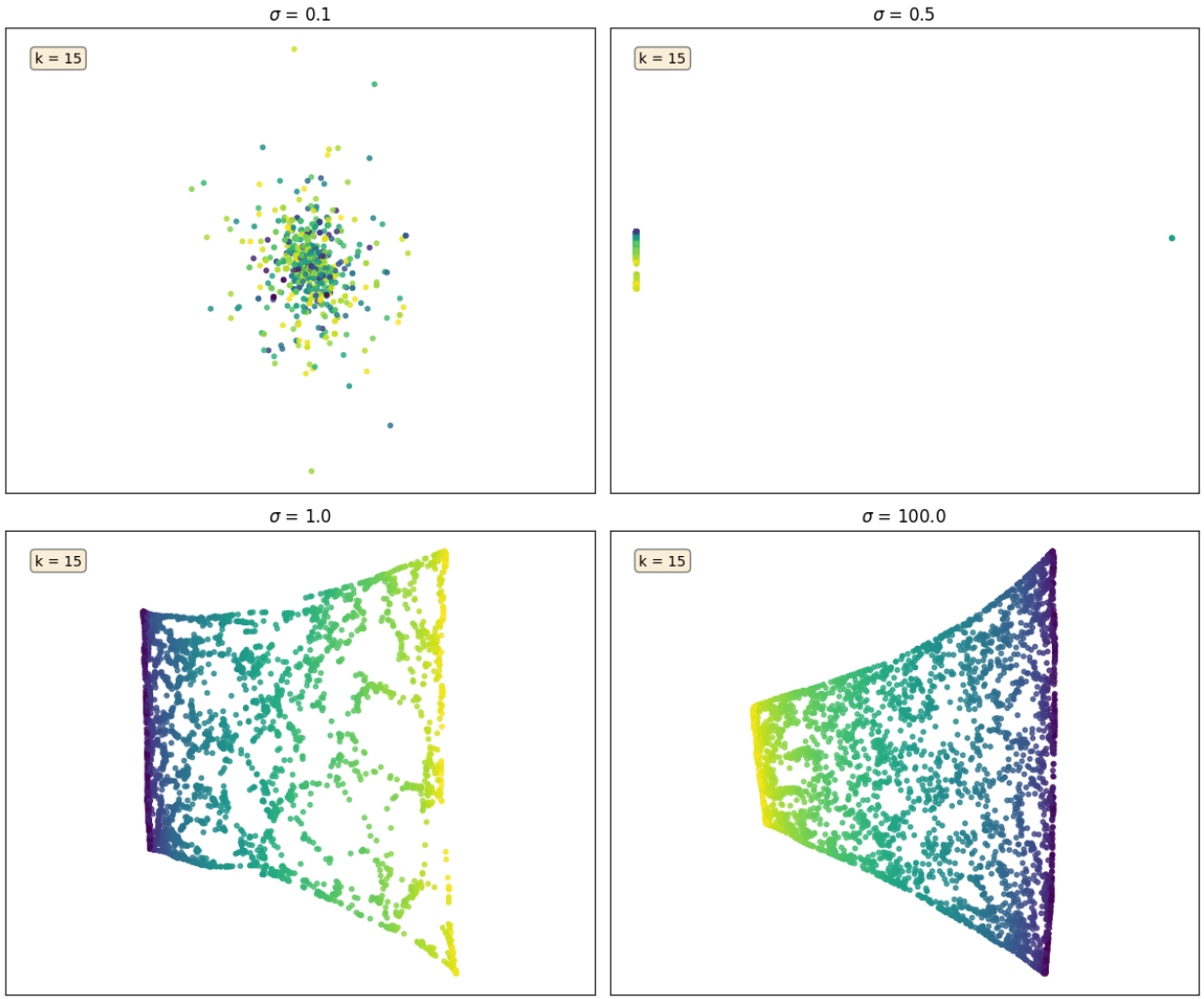


Figure: Effect of Varying Sigma (Kernel Bandwidth) on the Embedding

# PCA vs Laplacian Eigenmaps on Swiss Roll

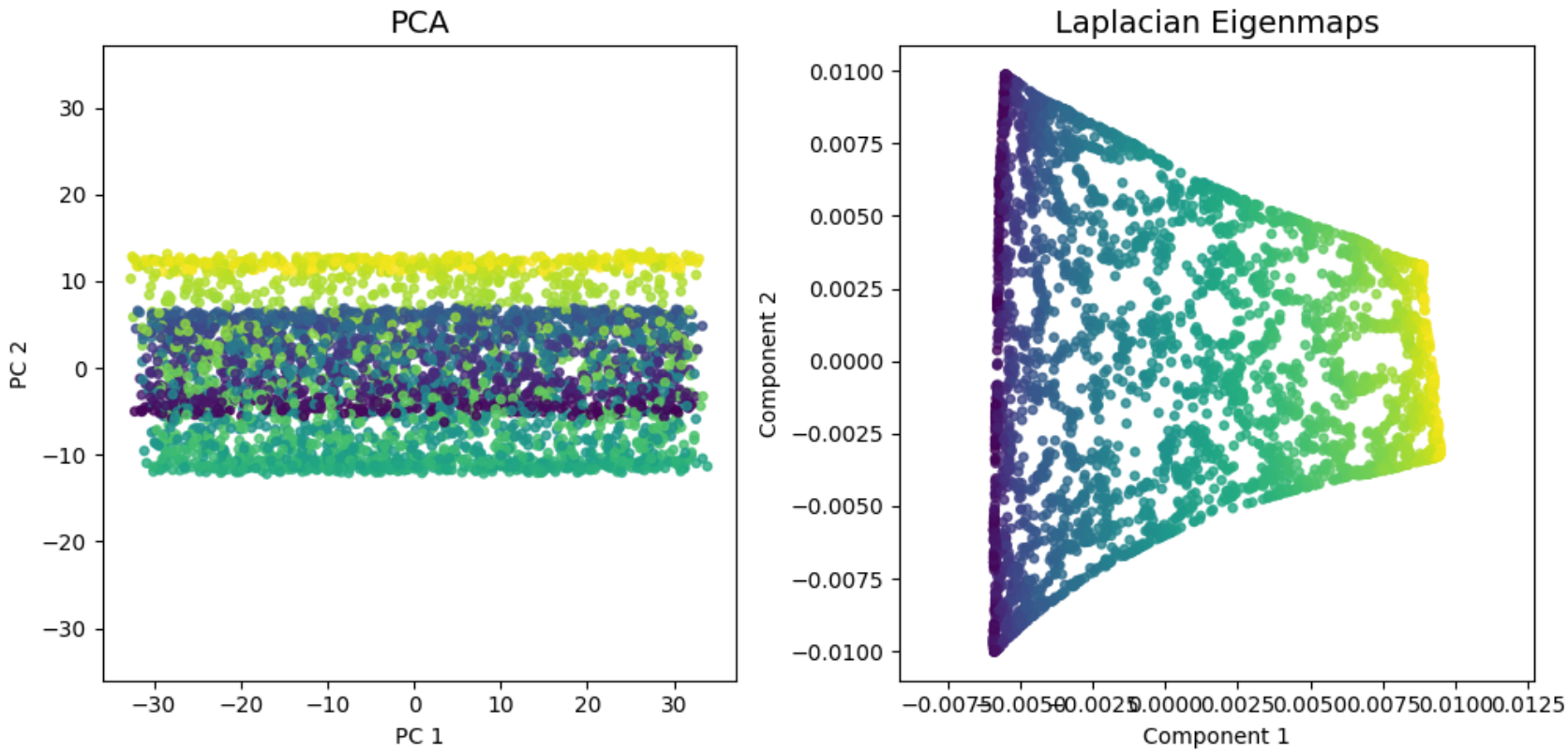


Figure: Comparison of PCA and Laplacian Eigenmaps on the Swiss Roll Dataset

# Mammoth Dataset

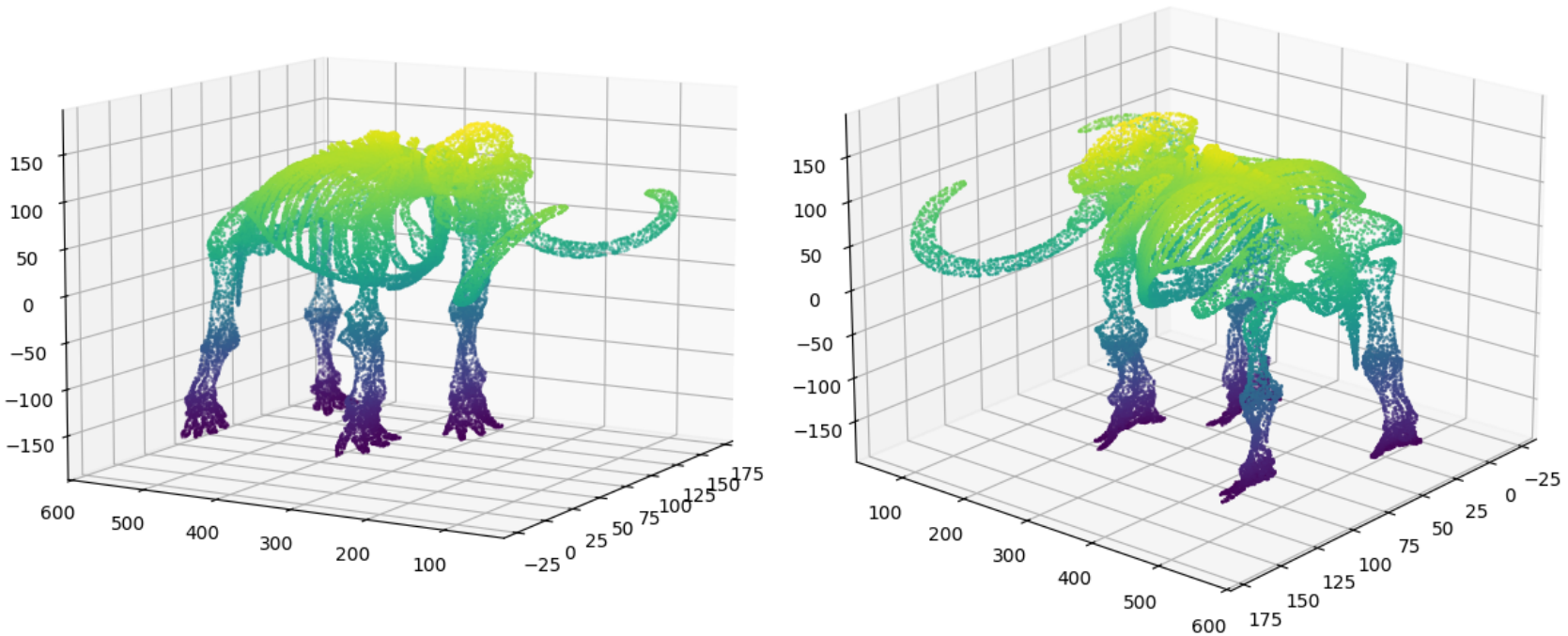


Figure: 3D Mammoth Dataset Visualization

# Laplacian Eigenmaps Embedding of Mammoth Dataset

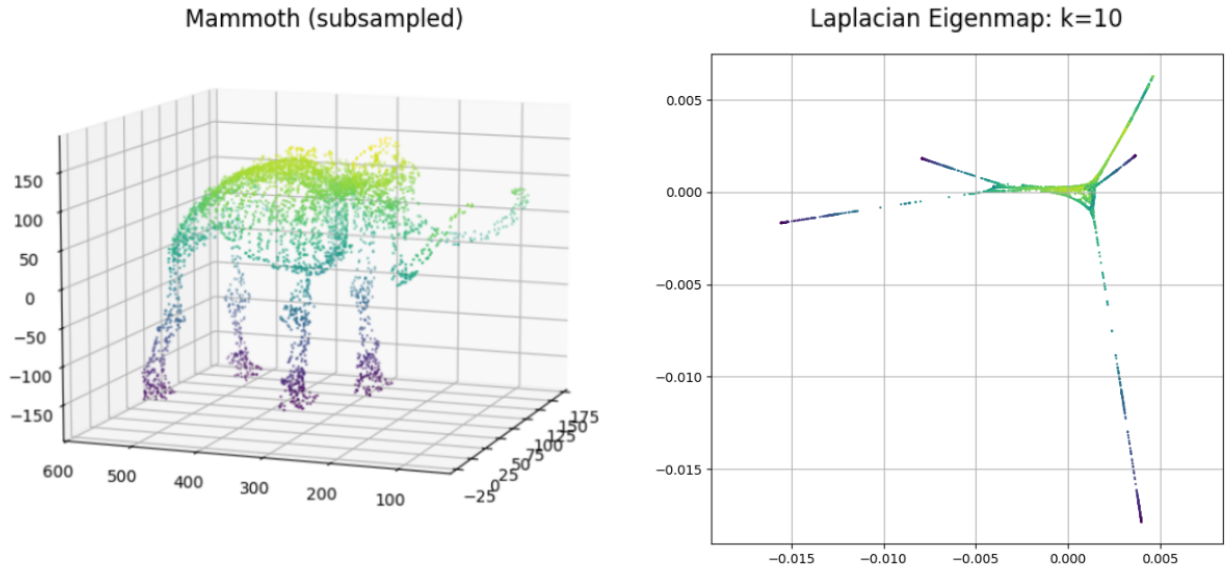


Figure: Laplacian Eigenmaps Embedding of the Mammoth Dataset

## Observation

The effect shown on the picture is called the starfish effect. It reflects perfectly how the Laplacian Eigenmaps preserves connectivity, but it fails to preserve global structure.

# Laplacian Eigenmaps as UMAP initialization

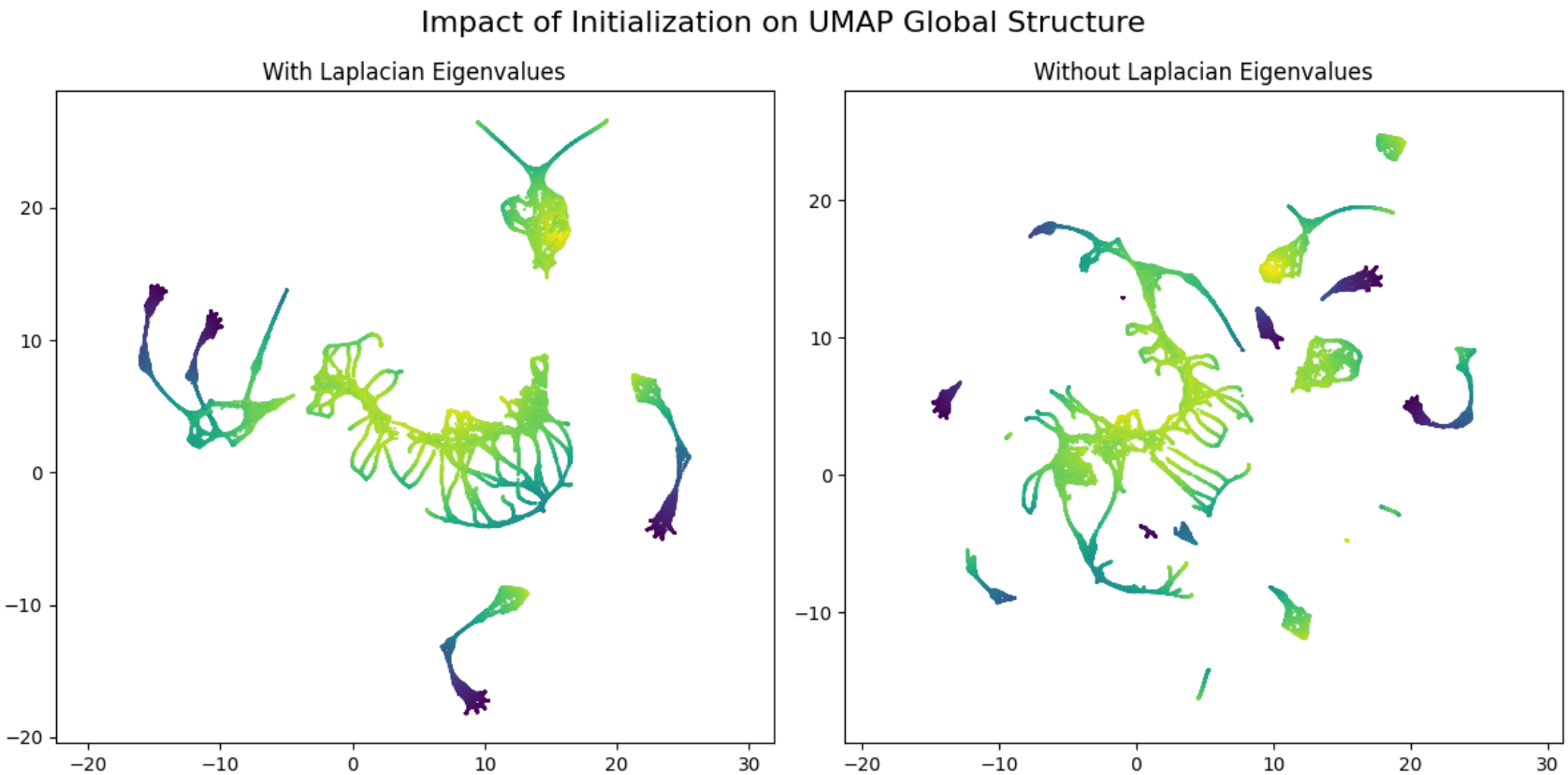


Figure: UMAP Embedding of the Mammoth Dataset with Laplacian Eigenmaps Initialization and Random Initialization