# Manifold Learning

## Parallel Transport Unfolding

Hugo Bouigeon - Paul Woringer

Dec. 2022

# Table of contents

# Introduction

Representing high dimensional data in lower dimensional spaces has many applications, ranging from visualisation to data preprocessing for machine learning algorithms. Many linear solutions have been proposed for this problem, including the Principal Component Analysis (PCA) algorithm which can be used to project data in lower dimensions while maintaining maximal variance. However some data sets are intrinsically non-linear, in that they do not lie on affine subspaces. In these cases, linear techniques are often inadequate and unable to reflect the true structure of the data.

Manifold learning techniques are very powerful alternatives to linear dimensionality reduction algorithms that may solve this issue in some cases. They rely on the hypothesis that the data lies on a **manifold**: a subspace of the high dimensional space that can locally be approximated by a tangent linear subspace. Isomap [2] is the most well-known of these algorithms, and many implementations are available in largely distributed libraries like Scikit-Learn. However, it can fail if the data sampling is too noisy or irregular.

Parallel Transport Unfolding [1] is a more stable alternative to the Isomap method. It is very good at conserving geodesic distances between data points. It is more robust to noisy data, and especially to irregular sampling (data sets with holes for example).

We implemented the PTU and Isomap algorithms and tested them on different point clouds to compare their performances and managed to obtain very satisfying results on a variety of datasets. The code is entirely our own production, and we only used functions from the Eigen and libigl libraries. We produced our own datasets by hand or with help from the Scikit-learn library in Python.

# Chapter 1

# Description of the PTU algorithm

The PTU algorithm is based on five different steps, which are the following:

1. Building a proximity graph.
2. Tangent space computation.
3. Discrete metric connections.
4. Unfolding for distance calculations.
5. Multidimensional scaling.

## 1.1 Proximity graph

The input of the algorithm is a plain point cloud with no connections. The first step of the PTU algorithm is to produce a proximity graph. There are two ways to do so: KNN and $\epsilon$-ball. In both cases, the length of an edge is the euclidean distance between its two extremities.

Using the k-nearest-neighbors algorithm to produce the proximity graph, for each vertex in the point cloud, we add an edge that goes to each of its $k$ nearest neighbors. Choosing $k = 4*d$, where $d$ is the number of dimensions of the final embedding, is empirically a good option, according to [1]. Note that the graph produced is directed, and that distances computed using it might be asymmetrical. See end of Section 1.4 for how to handle this issue.

Using the $\epsilon$-ball algorithm, we connect two points of the point cloud if their euclidean distance is lower than $\epsilon$. The graph produced is therefore clearly undirected. However, the result is highly dependent on the value of $\epsilon$, and the user has to tune this parameter to each situation (see Figure 2.3).

In any case, there is no guarantee that this graph is connected. [1] only considers the case of points that lie on a single connected manifold. However even in this case, if the sampling is too irregular, it is possible to have an unconnected graph. See end of Section 1.4 for how we chose to handle this issue.

## 1.2 Tangent space computation

Using the $K$ nearest neighbors of each point $x_i$ in the proximity graph, we can compute their local tangent space $T_i$ using the singular value decomposition of the associated local coordinate matrix of the set of neighbors. Usually one uses the same amount $k$ of neighbors as in the

previous step, but it is also possible to select a greater amount $K \geq k$ of neighbors to compute the tangent spaces. It is recommended by [1] to use higher values of $K$ when the dataset is very noisy.

## 1.3 Discrete metric connections

Once tangent spaces have been computed for each point of the dataset as explained in Section 1.2, we compute discrete metric connections between neighboring points on the proximity graph. If $x_i$ and $x_j$ share an edge on the proximity graph, we define:

$$R_{i,j} = \text{argmin}_{R \in O(d)} ||T_i - T_j R||^2$$

This corresponds to the change of base between the two tangent spaces $T_i$ and $T_j$.

To compute $R_{i,j}$, we use the formula $R_{i,j} = VU^\top$ where $V$ and $U$ come from the Singular Value Decomposition $T_i^\top T_j = U\Sigma V^\top$.

## 1.4 Unfolding for distance calculations

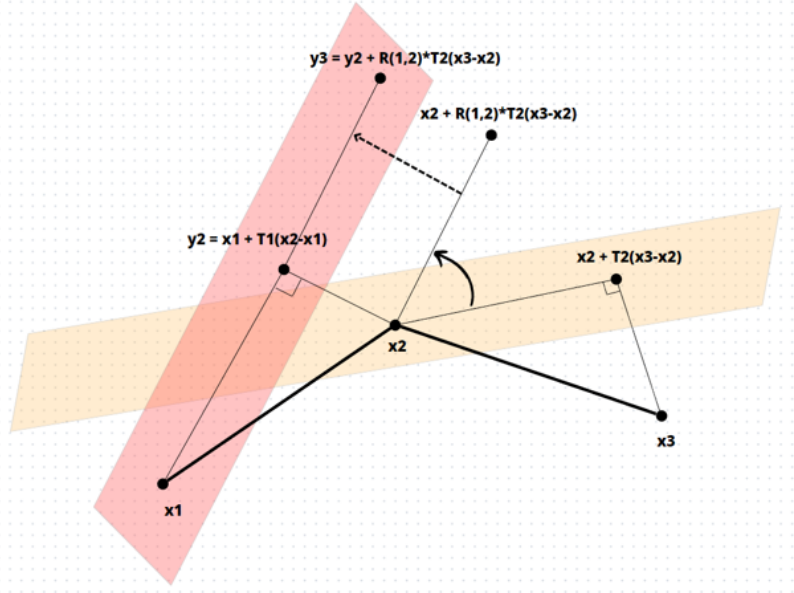



Figure 1.1: Sketch of the unfolding mechanism.

To compute the geodesic distance between two points $x_0$ and $x_n$, we first use the Dijkstra algorithm to find the shortest path from $x_0$ to $x_n$. Then, as illustrated in Figure 1.1, if $x_{n-1}$ is the previous point in the path, we project the vector $e_{n-1} = x_n - x_{n-1}$ in the tangent space of $x_{n-1}$ and rescale it to get:

$$v_{n-1} = ||e_{n-1}|| \frac{T_{n-1} e_{n-1}}{||T_{n-1} e_{n-1}||}$$

Then we use the $R_{k,k+1}$ matrices iteratively to 'rotate' this vector and map it to $T_0$, the tangent space of $x_0$. There we simply add it to $y_{n-1}$, the projection of $x_{n-1}$ via unfolding in $T_0$ to obtain $y_n$. Finally we compute the euclidean distance between $y_0 = 0$ and $y_n$ and use the square of this value in the distance matrix.

The only difference between Isomap and PTU is that Isomap simply uses the length of the Dijsktra path as the geodesic distance between the two points instead of performing this unfolding step.

If the graph is not connected, and we cannot find a path from point $x_0$ to $x_n$, we decided to simply use the Euclidean distance between the two points. This is certainly not ideal, but it yielded coherent results in our tests.

As mentioned in Section 1.1 the proximity graph can be asymmetrical, and the computed distance matrix might also be asymmetrical. To avoid that, we simply apply the $D \mapsto \frac{1}{2} * (D + D^\top)$ transformation to the distance matrix.

## 1.5 Multidimensional scaling (MDS)

Using the distance matrix $D$ computed in Section 1.4, we can compute the Gram matrix $G$ associated with it:

$$G = -\frac{1}{2}(I_n - \frac{1}{n}J_n)D(I_n - \frac{1}{n}J_n)$$

where $J_n$ is the $n$ by $n$ matrix filled with ones. We then compute the $d$ greatest eigenvalues of $G$ and their associated eigenvectors. The new embedding space of the dimensionally reduced vectors is then defined by the matrix:

$$Z = \sqrt{\Lambda_d} Q_d^\top$$

where $\Lambda_d$ and $Q_d$ are respectively the truncated matrices of the $d$ greatest eigenvalues and eigenvectors of $G$. This concludes the PTU algorithm.

# Chapter 2

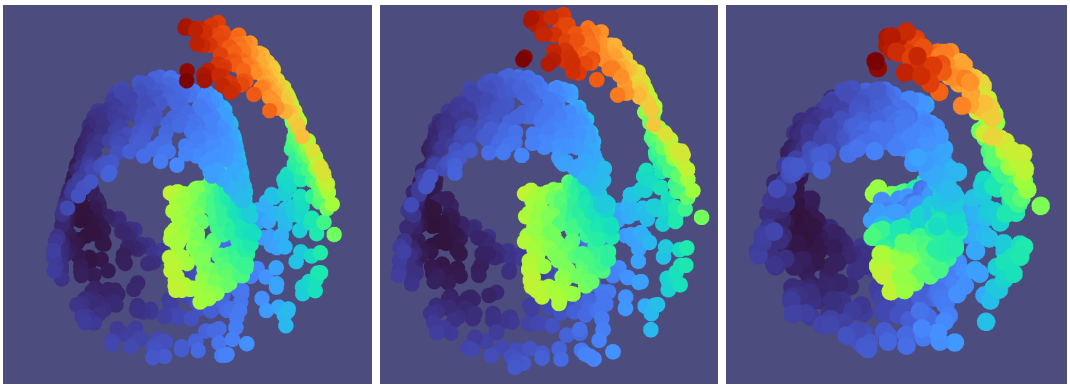
# Results

This chapter showcases the results of the above described algorithm we implemented. For visualization purposes, we only display the results of Isomap and PTU on 3D point clouds, with the resulting embeddings in 2D. However these algorithms and our implementation of them are able to function in higher dimensions. For the figures in 3D, the color of the points is determined by their distance to an arbitrarily-chosen point in dark purple (points further away progressively turn green, then yellow, and finally red).

## 2.1 Parallel Transport Unfolding Results

### 2.1.1 Influence of noise

Parallel Transport Unfolding is quite robust to noise, and can be adapted by choosing adequate values of $k$ and $K$ to withstand noise even more, as explained in Section 1.2. However, excessive levels of noise will still cause results to worsen somewhat, as shown in Figure 2.1. This is natural, as the hypothesis that the data lies on a manifold is less and less true as noise increases, and the structure itself of the data becomes affected by the noise.



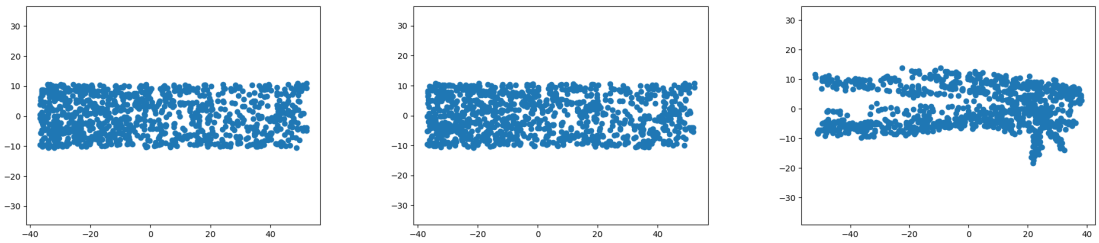


Figure 2.1: PTU results on the swiss roll dataset with standard deviation of the Gaussian noise set to 0.0, 0.2 and 0.5.

### 2.1.2 Use of $\epsilon$ ball method

As mentioned in Section 1.1, we can use the $\epsilon$-ball method instead of KNN to compute the adjacency graph. In some cases, like in Figure 2.2, this variant performs better.

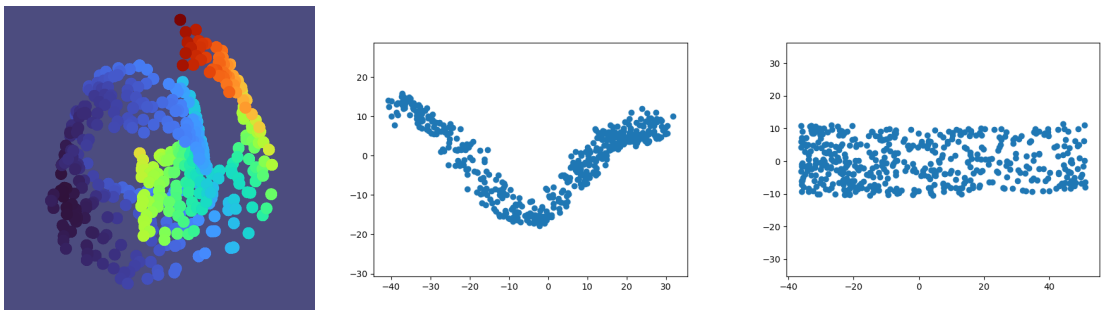


Figure 2.2: PTU results for swiss roll dataset with low density and highly irregular sampling using the KNN method (center) and $\epsilon$ ball method (right).

One disadvantage to the $\epsilon$-ball method though is that the $\epsilon$ parameter can change the result of the unfolding significantly. It is primordial to set it in such a way, that it is great enough to recognize local neighbors of each data point. However, $\epsilon$ also has to be small enough so that it does not misjudge data-points as neighbors when they are not and create bridges between parts of the dataset that should be far apart from one another. This makes this method somewhat hard to automate.

As you can see in 2.3, inappropriate values for $\epsilon$ result in very distorted embeddings.

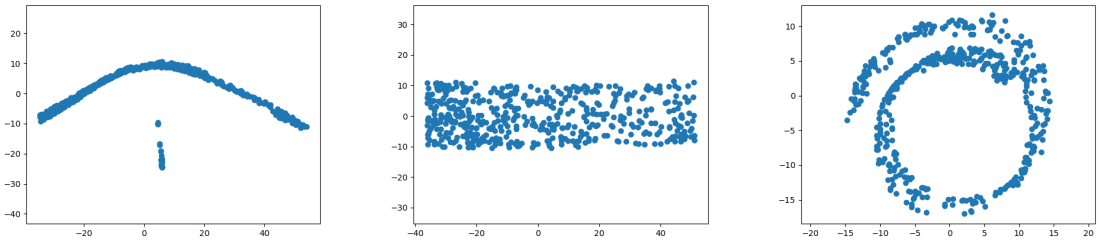


Figure 2.3: Results for the swiss roll data set with the $\epsilon$ parameter set to 3.0 (left), 4.0 (center), and 8.0 (right).

## 2.2 Comparing Isomap to PTU

We implemented both the Isomap and the Parallel Transport Unfolding methods described in [1]. We could then compare both methods. Figure 2.4 shows side-by-side results of Isomap and PTU.

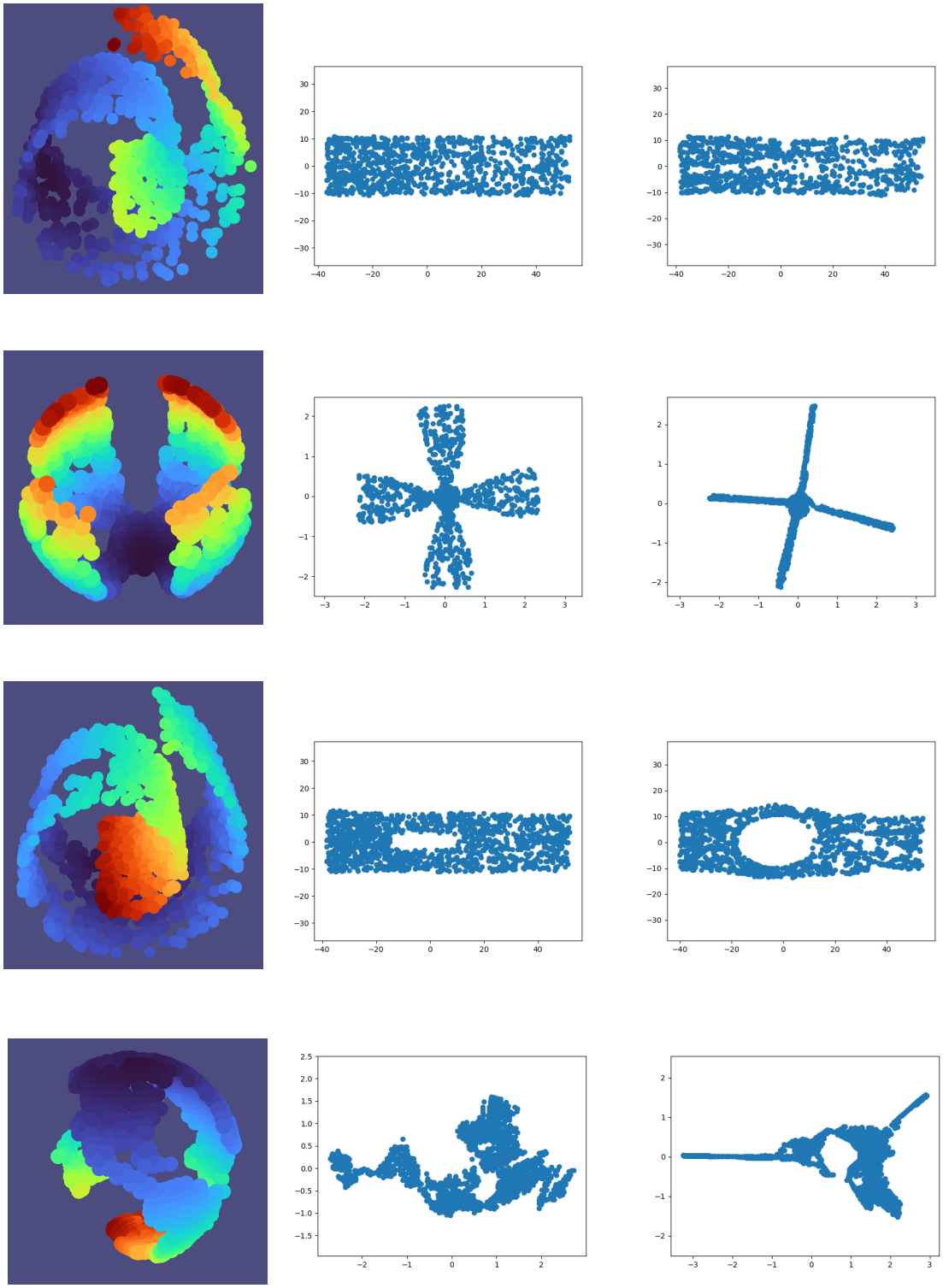


Figure 2.4: Comparing the PTU method (center) with the Isomap method (right) on different 3-dimensional datasets (left).

As expected, PTU yields better results on most of the datasets we tested. In particular, it is known that Isomap tends to overestimate distances around holes in datasets, which can be seen in our results (third row of Figure 2.4). Only the very noisy swiss roll dataset turned out better with the Isomap method as shown in 2.5, whether we used the KNN or $\epsilon$-ball method. This may be due to the fact that we consistently used $K = k$ in our tests, and increasing the value of $K$ could have solved this.

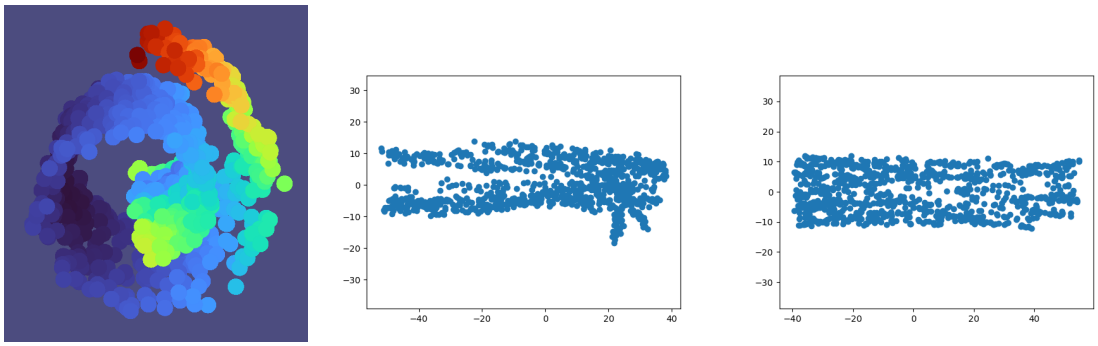



Figure 2.5: Comparing the PTU method (center) with the Isomap method (right) on the noisiest swiss roll dataset (standard deviation Gaussian noise set to 0.5).

## Conclusion

We implemented two manifold learning methods: Parallel Transport Unfolding and Isomap. Both are capable of creating quasi-isometric embeddings of higher dimensional data. Isomap is a method that is slightly easier to implement, but it shows some shortcomings on datasets containing holes or with unconventional layouts such as the flower dataset or earth dataset we made. PTU yields significantly better results in these cases. On top of that, Parallel Transport Unfolding is also quite robust to noise and allows for the tweaking of parameters to improve results. It also conserves the same asymptotic time complexity as the Isomap method.

Shortcomings of both Isomap and PTU are that they are conceived solely to treat data which takes the shape of a manifold. However, in real life, data does not always verify this condition. Interlacing rings for example could not be represented correctly with Manifold Learning. Hence other dimension reduction techniques will always be necessary. There currently is no superior technique that performs best in each possible scenario, and we have shown in this project that PTU is an excellent option for data sets that lie on manifolds.

# Bibliography

[1] Max Budninskiy et al. "Parallel Transport Unfolding: A Connection-based Manifold Learning Approach". In: (2018). DOI: 10.48550/ARXIV.1806.09039. URL: https://arxiv.org/abs/1806.09039.

[2] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". In: *Science* 290.5500 (Dec. 2000), pp. 2319–2323. DOI: 10.1126/science.290.5500.2319. URL: https://doi.org/10.1126/science.290.5500.2319.