Diffusion Maps for Dimensionality Reduction

Introduction

In this paper, we discuss diffusion maps, a technique for dimensionality reduction. The goal of dimensionality reduction is to compute a low-dimensional representation of high-dimensional data, as it often difficult to work with the latter directly. The underlying structure of the data should be maintained in the low-dimensional representation. Diffusion maps accomplish this by running a diffusion process that measures the similarity between points at different time steps.

We implemented two diffusion map algorithms. The first is a basic algorithm that uses a Gaussian kernel and computes the normalized graph Laplacian. The second is an extension of the first that approximates the Fokker-Planck diffusion (Coifman and Lafon). We compare the two algorithms with the common dimensionality reduction technique Principal Component Analysis (PCA) on four three-dimensional datasets. Results are given in both one and two dimensions.

The Basic Diffusion Map

The first algorithm is the most basic form of a diffusion map. The version described here most closely follows that of de la Porte et al. For any two points x and y in our dataset, we define a Gaussian kernel function by

$$k(x, y) = \exp\left(-\frac{||x - y||^2}{\epsilon}\right),$$

where $||x - y||$ is the standard Euclidean distance between x and y.

Notice that this function is symmetric and nonnegative. The kernel function is a representation of the distance between two points, where the parameter ϵ defines what is considered "close". A smaller value of ϵ means that points must be closer together to be considered "close", and the opposite is true for a larger value of ϵ. This makes the effectiveness of the algorithm heavily dependent on the choice of ϵ, and different values of ϵ are appropriate for different datasets.

It became clear during implementation that a single static value of ϵ would not work equally well for all of the test datasets. Instead, the basic algorithm uses a variation of the median heuristic proposed by Schclar:

$$\epsilon = \text{median}\{||x - y||^2\},$$

where the median is taken over all points x and y in the dataset. In other words, the median squared distance between all pairs of points is used as a measure of "closeness" that a given pair of points can be compared to.

We compute the kernel function for all pairs of points x_i and x_j in the dataset. This forms the symmetric kernel matrix K, where the i, j entry is

$$K_{ij} = k(x_i, x_j).$$

Next, we row-normalize the kernel matrix to create the diffusion matrix P, where

$$P_{ij} = \frac{1}{\sum_j K_{ik}} K_{ij}.$$

Since the rows of P sum to 1, the entries can be considered as probabilities. Entry P_{ij} is the probability of jumping from point x_i to point x_j. These probabilities are greater for points that are close together because of how the kernel function is defined. When considered this way, P is the transition matrix of a Markov chain that defines a random walk on the dataset. Similarly, if the dataset is conisidered as a graph, then P is a form of the normalized graph Laplacian (Coifman and Lafon).

An interesting property of the diffusion matrix involves powers of the matrix. It is shown by de la Porte et al that the entries P^t_{ij} are the probabilities of paths of length t from x_i to x_j. This gives the ability to run the diffusion process forward in time, revealing information about the data at larger scales. For the purposes of this project, we are interested in smaller scale relationships, so we only use $t = 1$. Larger values of t result in the output data becoming clustered together too much (to a single point for large enough t), as the algorithm is using too large of a scale to identify meaningful relationships.

We can now use the probabilities P_{ij} to compute the diffusion distance between two points:

$$D(x_i, x_j) = \left(\sum_k |P_{ik} - P_{kj}|^2 \right)^{1/2}.$$

The diffusion distance measures the probability of jumping from one point to another through all possible paths. This provides a measure of similarity between points that takes into account the underlying structure and connectivity of the data, as opposed to Euclidean distance which does not.

The final step of the algorithm is to compute the eigenvalues and eigenvectors of the diffusion matrix P. Let $\lambda_1, \dots, \lambda_n$ be the eigenvalues, and let $\psi_1, \dots, \psi_n$ be the corresponding eigenvectors. The diffusion map relates to each point x_i of the original dataset a new mapped point

$$x'_i = (\lambda_1 \psi_1(i), \dots, \lambda_n \psi_n(i)),$$

where $\psi_j(i)$ denotes the i-th element of ψ_j. It is shown by Coifman and Lafon that the Euclidean distance between x'_i and x'_j is equal to the diffusion distance between x_i and x_j. Thus the points in the diffusion mapped space are organized according to the diffusion distance of the original points.

To reduce the data to $m < n$ dimensions, we retain the m elements of x'_i corresponding to the m dominant eigenvalues, for each point x'_i. This results in an m-dimensional dataset where the Euclidian distance between mapped points approximates the diffusion distance between the original points.

The Modified Diffusion Map

The modified diffusion map is an extension of the basic diffusion map, intended to improve its performance on the test datasets. The major difference is that an extra normalization step is added in between the computation of the kernel matrix K and the normalization to the diffusion matrix P. This leads to an approximation of the Fokker-Planck diffusion.

After we compute the kernel matrix K as in the basic diffusion map, we create a new normalized kernel matrix M. The i, j entry of M is

$$M_{ij} = \frac{1}{(r_i r_j)^\alpha} K_{ij},$$

where r_k is the sum of the kth row of K. We then row-normalize M to create P in the same way as in the basic diffusion map (with M in place of K), and the remaining steps proceed indentically.

Coifman and Lafon show that, as the parameter of the kernel function $\epsilon \to 0$, the infinitesimal generator of the Markov chain defined by P becomes

$$\mathcal{L}f = \Delta\phi - \frac{\Delta(q^{1-\alpha})}{q^{1-\alpha}}\phi,$$

where q is the true probability density and $\phi = fq^{1-\alpha}$. Different values of α lead to different types of diffusions.

Notice that if $\alpha = 0$, then $M = K$. Hence there is no difference from the basic diffusion map in this case. The generator becomes

$$\mathcal{L}f = \Delta\phi - \frac{\Delta q}{q}\phi.$$

For the modified diffusion map, we use $\alpha = 1/2$. The resulting generator is

$$\mathcal{L}f = \Delta\phi - \frac{\Delta(\sqrt{q})}{\sqrt{q}}\phi,$$

from which we can derive the Fokker-Planck equation (forward Kolmogorov equation). Because of this, the algorithm with $\alpha = 1/2$ can be used to analyze a stochastic dynamical system. We chose to implement this version in order to test whether the Fokker-Planck approximation allows the algorithm to better handle noise and stochasticity in the data.

The other major change to the modified diffusion map algorithm is the value of the parameter ϵ. Since the Fokker-Planck approximation improves as $\epsilon \to 0$, it is beneficial to use a smaller value of ϵ than the median heuristic while still having the parameter depend on the specific data. To this end, we use a variation of the max-min heuristic also proposed by Schclar:

$$\epsilon = \max_i \min_j \|x_i - x_j\|^2.$$

In other words, we find the squared distance of each point to its closest neighbor and then use the maximum of those squared distances as a measure of closeness.

Implementation

We implemented the algorithms, as well as scripts for generating and plotting the datasets, in Python. The PCA algorithm uses the skikit-learn library's PCA implementation, while the two diffusion map algorithms are implemented using only standard numeric libraries such as numpy and scipy.

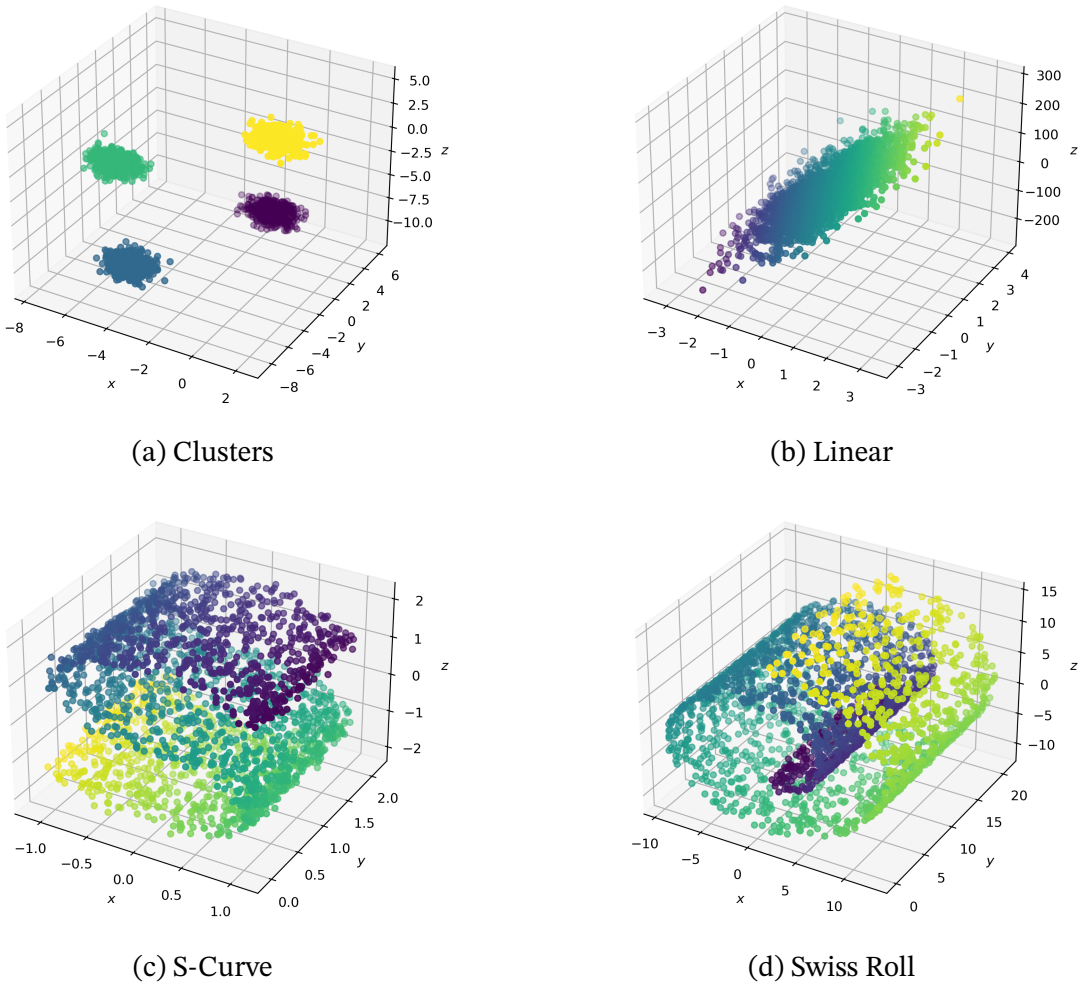
One implementation detail of note is the computation of the eigenvalues and eigenvectors of P in both diffusion map algorithms. Although the kernel matrix is symmetric, the row-normalized diffusion matrix P is not. However, eigenvalue algorithms specificically for symmetric matrices, such as scipy's `eigh`, are faster and more accurate than algorithms for general matrices. Thus the diffusion map algorithms follow a process outlined by Schclar, which computes a symmetric matrix A that is similar to P. The algorithms then transform the eigenvectors of A back into eigenvectors of P.

To test the algorithms, we generated and plotted four three-dimensional datasets, which are easily visualizable. Each dataset contains 2500 points. The algorithms were run on each dataset twice: once to reduce the data to two dimensions and once to reduce the data to one dimension. In each dataset, points were labeled with colors. Each output point of the algorithms was labeled with the same color as the corresponding input point in order to easily visualize how the points are mapped. We then plotted the output datasets.

The datasets are:

- Clusters: Points are grouped into four clusters, with colors corresponding to each cluster.
- Linear: Points are arranged roughly on a two-dimensional plane, with colors corresponding to position on the plane.
- S-Curve: Points are arranged on an s-curve, with colors corresponding to parametric position on the curve.
- Swiss Roll: Points are arranged on a swiss roll shape, with colors corresponding to parametric position on the curve.

Each dataset has some underlying noise, so the points are not positioned exactly according to a curve or structure. Figure 1 shows the datasets.



(a) Clusters

(b) Linear

(c) S-Curve

(d) Swiss Roll

Figure 1: Datasets

Results

Before examining the results, we make two observations. First, it should be easier for the algorithms to identify two-dimensional structure than one-dimensional structure, as the former is a less significant reduction. Second, PCA is a linear technique while the diffusion maps are nonlinear, so the diffusion maps should be more effective at handling nonlinearity and noise. Plots of the one-dimensional and two-dimensional output of each algorithm on each dataset are shown at the end of the document.

Figure 2 shows the clusters output. PCA and the basic diffusion map perform as expected in both dimensions. Interestingly, the clusters in the two-dimensional basic diffusion map output are more skewed than those in the PCA output, which are more circular. This could be because the basic diffusion map is able to detect some skewness in the original data.

On the other hand, the modified diffusion map seems to "over reduce" the data, as the clusters are mapped into single points each. Furthermore, the purple and yellow clusters, which

are the closest together in the original dataset, overlap almost completely. This suggests that the modified diffusion map is too powerful for the relatively simple structure of the clusters dataset.

Figure 3 shows the linear output. The differences between PCA and the diffusion maps are more significant for this dataset. In two dimensions, PCA outputs a structure similar to a flattened version of the original data. The diffusion maps, however, reduce the data further into a single curve while maintaining the order from top to bottom of the original data. Depending on the goal of the reduction, there could be reasons to prefer one or the other. The diffusion maps provide a stronger two-dimensional reduction, closer to the one-dimensional reduction, at the cost of losing some of the original shape of the data.

Figure 4 shows the s-curve output. PCA and the basic diffusion map produce similar results. The shape of the S is preserved in two dimensions, while both algorithms generally preserve the middle of the curve in one dimension but have trouble at each end. In comparison, the one dimensional output of the modified diffusion map is significantly improved. The modified diffusion map's two-dimensional output is a similar situation to the linear dataset. It provides a greater reduction than the other algorithms, to a parabola-like curve, but loses the S shape.

Finally, Figure 5 shows the swiss roll output. The swiss roll can be thought of as a more extreme form of the s-curve, and the results reflect that. PCA and the basic diffusion map preserve the spiral shape in two dimensions but struggle in one dimension. The modified diffusion map, meanwhile, performs well in one and two dimensions, with the two-dimensional reduction having the same benefits and drawbacks as for the s-curve.

Conclusion

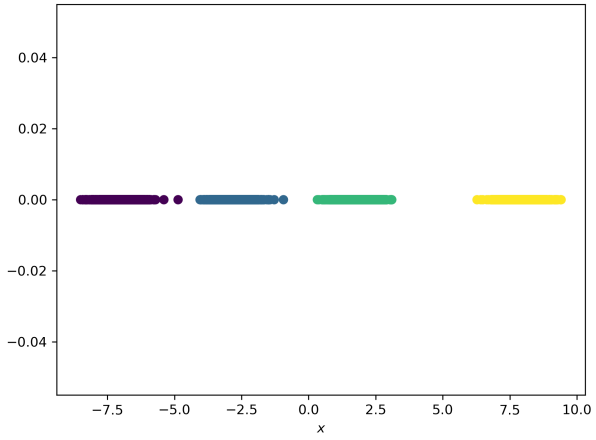
In conclusion, diffusion maps are an effective technique for dimensionality reduction. Comparisons to PCA are favorable, as the diffusion map algorithms perform similarly or better on each dataset. The modified diffusion map algorithm provides the strongest reduction of the three, which can likely be attributed to the smaller value of ϵ and the Fokker-Planck diffusion's ability to handle noise.

The strength of the reduction is an interesting aspect to consider. The modified diffusion map on the s-curve and swiss roll datasets, and both diffusion maps on the linear dataset, reduce the data to a single curve in two dimensions rather than a flattened shape. As noted, this has advantages and disadvantages depending on the goal of dimensionality reduction. An extreme example is the modified diffusion map's performance on the clusters dataset, which demonstrates why a stronger reduction may not be preferred at all in some cases.

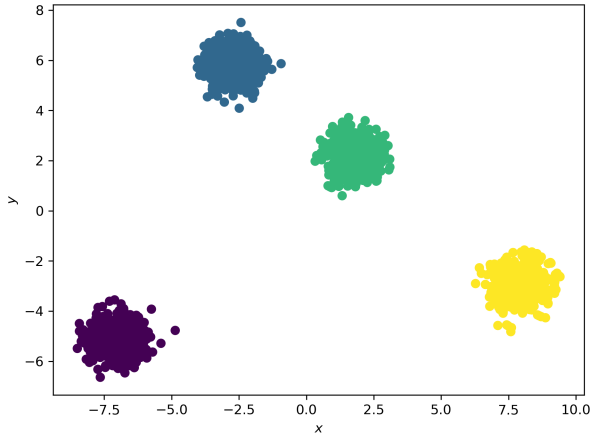
Lastly, as mentioned earlier, the performance of the diffusion map algorithms can be highly dependent on the value of the parameter ϵ. This is true for other parameters as well, including the time step t and the normalization power α. Different values of α for instance lead to approximations of different diffusions. Alongside the above discussion on reduction strength, this showcases the need to consider the structure of the dataset and goals of the analysis when implementing a diffusion map algorithm.

Bibliography

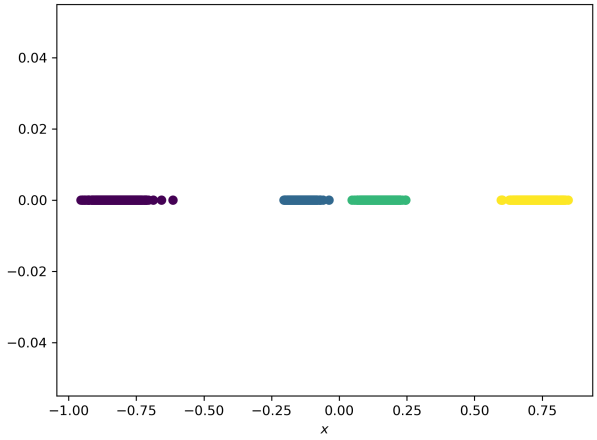
- Ronald R. Coifman, Stéphane Lafon, Diffusion maps, Applied and Computational Harmonic Analysis, Volume 21, Issue 1, 2006, Pages 5-30, ISSN 1063-5203, https://doi.org/10.1016/j.acha.2006.04.006. (https://www.sciencedirect.com/science/article/pii/S1063520306000546)
- de la Porte, J & Herbst, Ben & Hereman, Willy & van der Walt, Stéfan. (2008). An Introduction to Diffusion Maps (https://www.researchgate.net/publication/228567023_An_Introduction_to_Diffusion_Maps).
- Schclar, A. (2008). A Diffusion Framework for Dimensionality Reduction. In: Maimon, O., Rokach, L. (eds) Soft Computing for Knowledge Discovery and Data Mining. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-69935-6_13
- Boaz Nadler, Stéphane Lafon, Ronald R. Coifman, and Ioannis G. Kevrekidis. 2005. Diffusion Maps, spectral clustering and eigenfunctions of Fokker-Planck operators. In Proceedings of the 18th International Conference on Neural Information Processing Systems (NIPS'05). MIT Press, Cambridge, MA, USA, 955-962. (https://proceedings.neurips.cc/paper/2005/hash/2a0f97f81755e2878b264adf39cba68e-Abstract.html)
- Boaz Nadler, Stéphane Lafon, Ronald R. Coifman, Ioannis G. Kevrekidis, Diffusion maps, spectral clustering and reaction coordinates of dynamical systems, Applied and Computational Harmonic Analysis, Volume 21, Issue 1, 2006, Pages 113-127, ISSN 1063-5203, https://doi.org/10.1016/j.acha.2005.07.004. (https://www.sciencedirect.com/science/article/pii/S1063520306000534)
- S. Lafon and A. B. Lee, "Diffusion maps and coarse-graining: a unified framework for dimensionality reduction, graph partitioning, and data set parameterization," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 9, pp. 1393-1403, Sept. 2006, doi: 10.1109/TPAMI.2006.184. (https://ieeexplore.ieee.org/abstract/document/1661543)
- Ralf Banisch, Zofia Trstanova, Andreas Bittracher, Stefan Klus, Péter Koltai, Diffusion maps tailored to arbitrary non-degenerate Itô processes, Applied and Computational Harmonic Analysis, Volume 48, Issue 1, 2020, Pages 242-265, ISSN 1063-5203, https://doi.org/10.1016/j.acha.2018.05.001. (https://www.sciencedirect.com/science/article/pii/S1063520318301222)



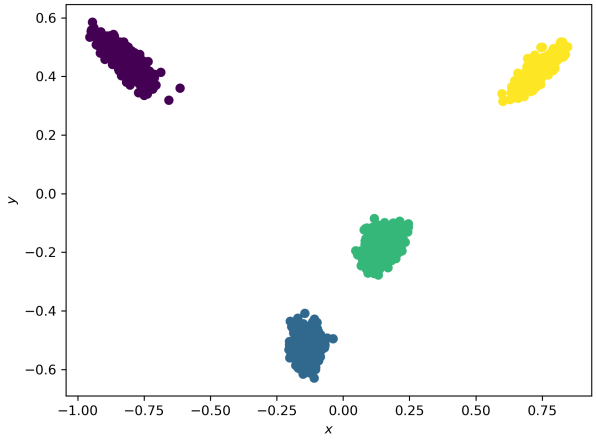
(a) 1D PCA



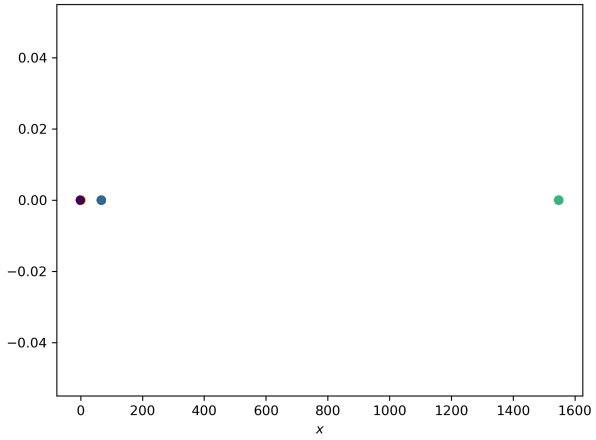
(b) 2D PCA



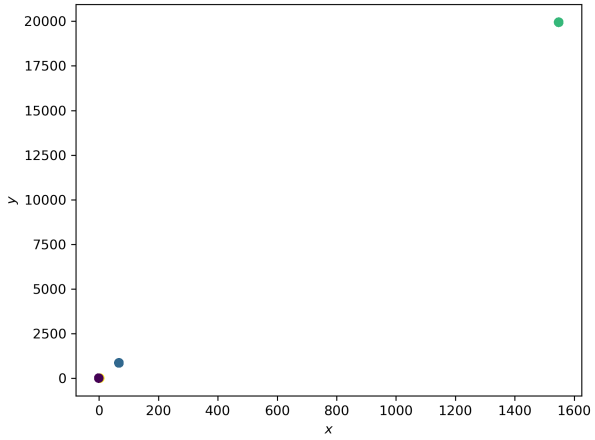
(c) 1D Basic Diffusion Map



(d) 2D Basic Diffusion Map

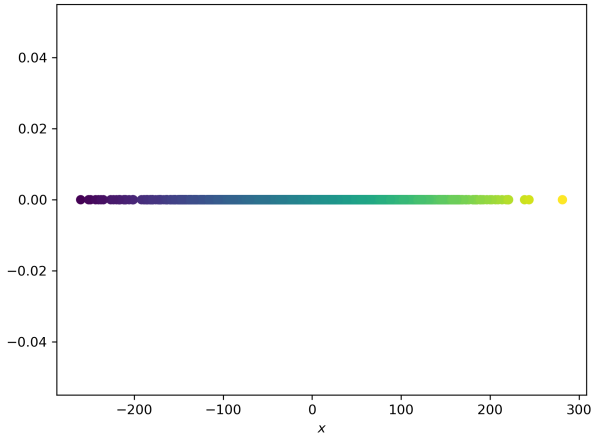


(e) 1D Modified Diffusion Map

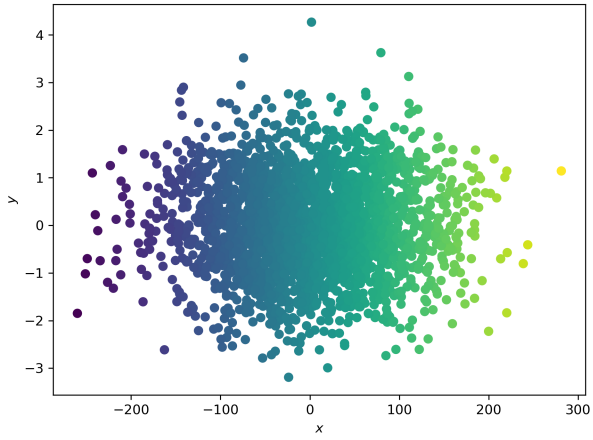


(f) 2D Modified Diffusion Map

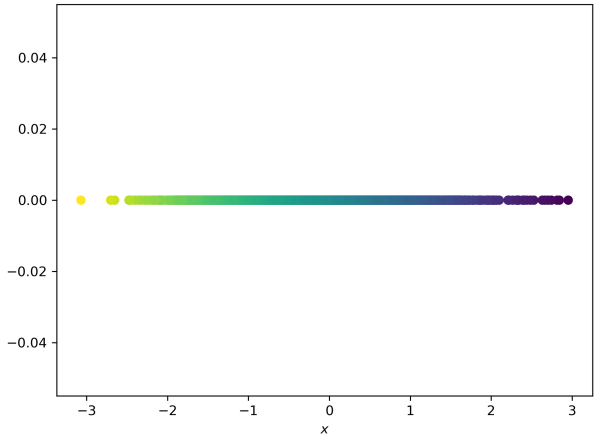
Figure 2: Clusters Output



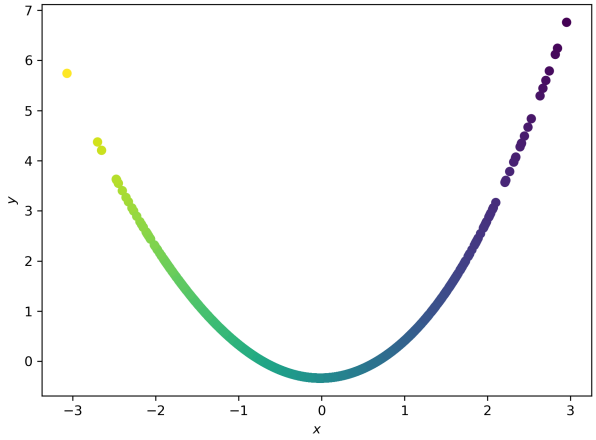
(a) 1D PCA



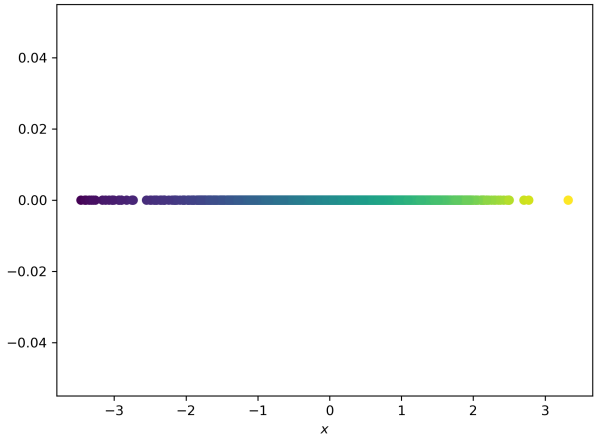
(b) 2D PCA



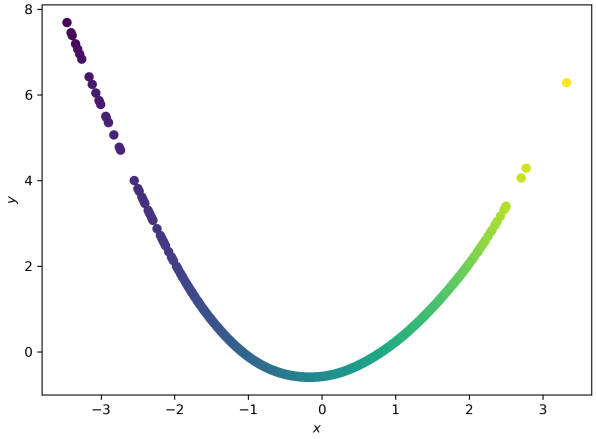
(c) 1D Basic Diffusion Map



(d) 2D Basic Diffusion Map

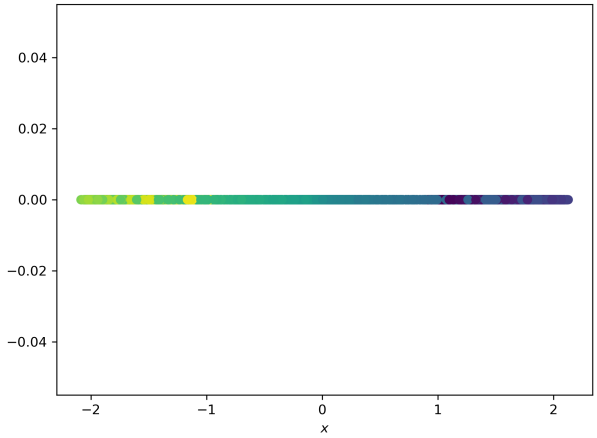


(e) 1D Modified Diffusion Map

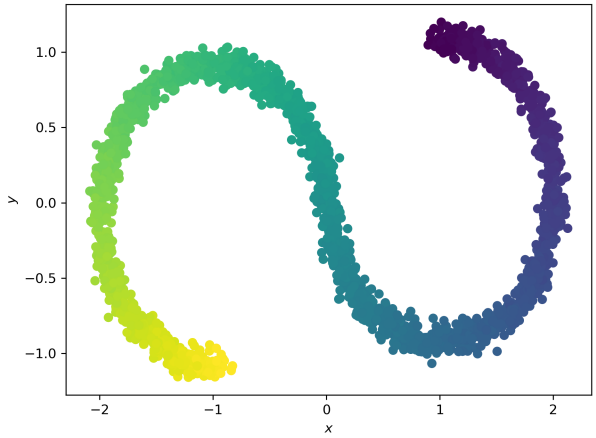


(f) 2D Modified Diffusion Map

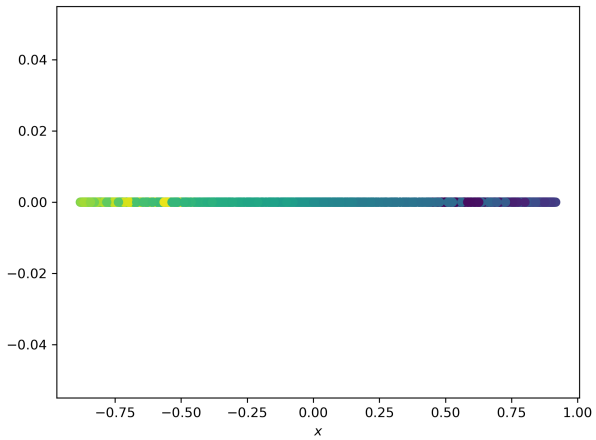
Figure 3: Linear Output



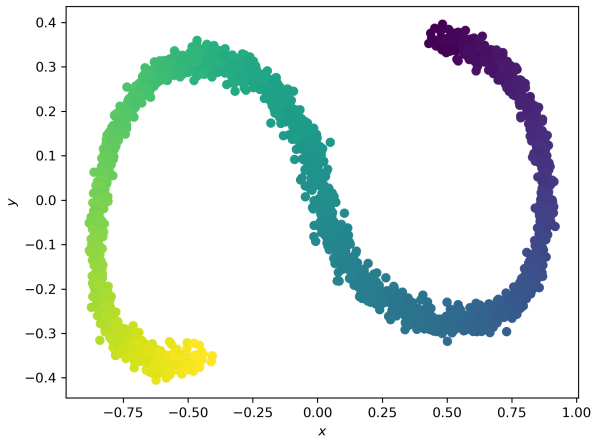
(a) 1D PCA



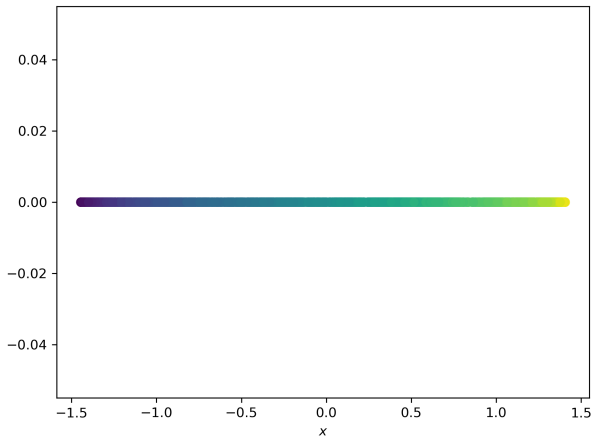
(b) 2D PCA



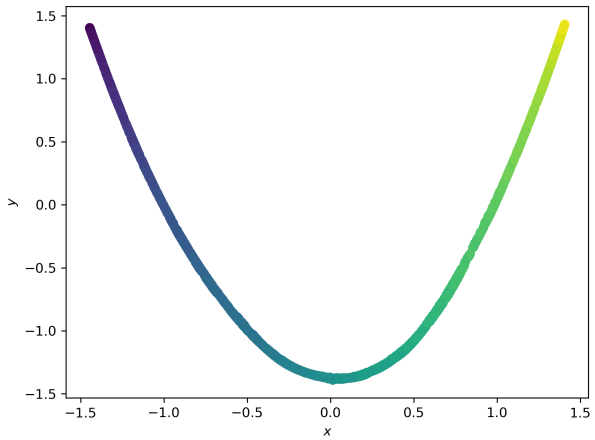
(c) 1D Basic Diffusion Map



(d) 2D Basic Diffusion Map

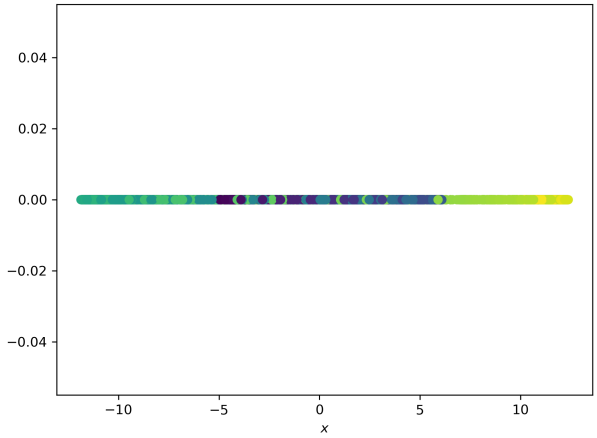


(e) 1D Modified Diffusion Map

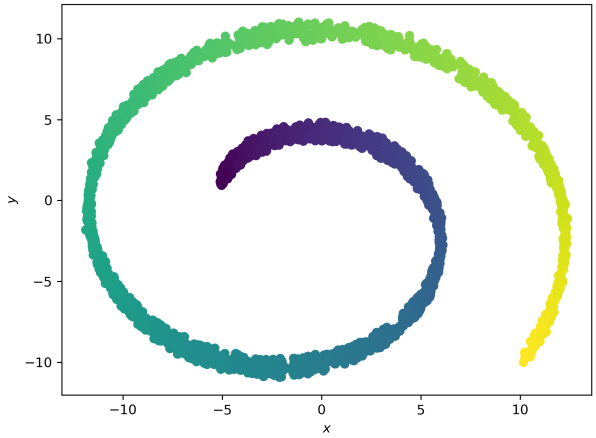


(f) 2D Modified Diffusion Map

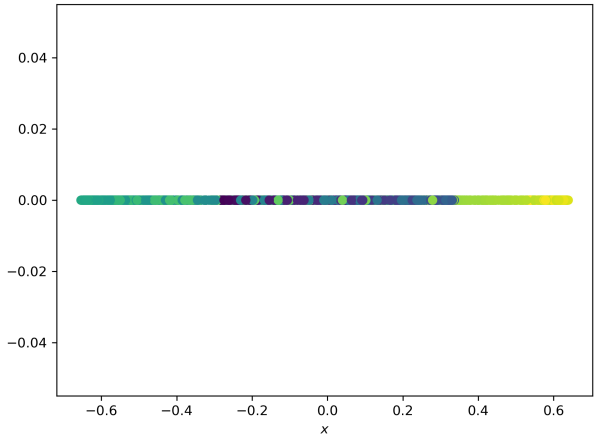
Figure 4: S-Curve Output



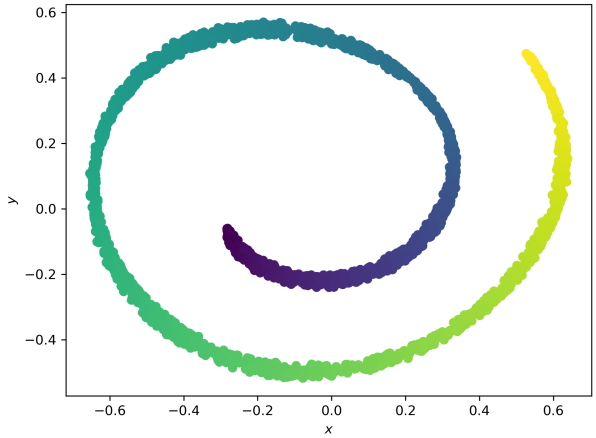
(a) 1D PCA



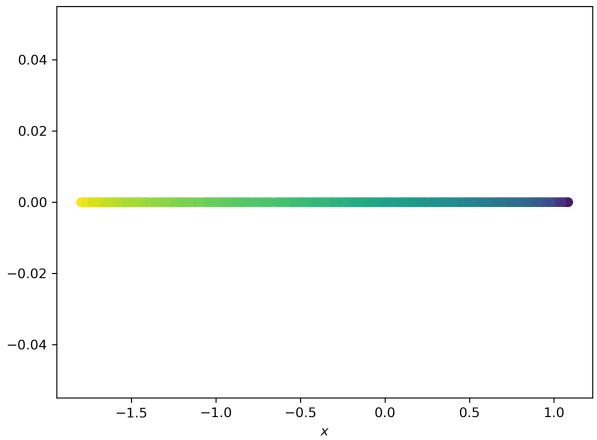
(b) 2D PCA



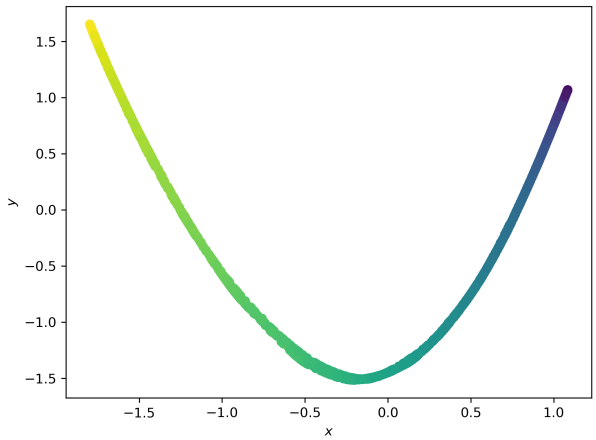
(c) 1D Basic Diffusion Map



(d) 2D Basic Diffusion Map



(e) 1D Modified Diffusion Map



(f) 2D Modified Diffusion Map

Figure 5: Swiss Roll Output