Time Series Analysis with Persistent Homology

Thesis submitted to

Prof. Dr. Roggenkamp, University of Mannheim

in partial fulfilment for the award of the degree of

Bachelor of Science

in Business Mathematics

By

Paul Gerhard

January 31, 2022

Contents

Abstract

In this Bachelor Thesis I present a method to analyze scalar time series. The method is based on homological algebra, persistent homology in particular. In persistent homology the goal is to obtain topological features of the space in which the data is distributed. Time Series, however, are not particularly suited for this kind of analysis since the space in which they lie is not interesting, topologically speaking. To make the space more interesting, one assumes that the underlying dynamic is in reality a higher dimensional system D and the time series t is simply a lower dimensional measurement of this system. Now one can apply Takens Embedding Theorem [1] to t and obtain a topologically much more interesting state space to analyze with persistent homology, whose dynamics are equivalent to those of the time series t.

After the state space D is reconstructed there are many ways to work with this space. In this work I follow the approach of Firas et al [5], who construct a network from a permutation sequence which is then analyzed with persistent homology.

Chapter 1

Introduction

Time series are present everywhere. From electrocardiograms which monitor heart rhythms to detect problems of the heart to financial time series which describe the supply and demand of a certain asset. Thus it is obvious that a lot of research goes into making the best predictions. The work that is done to analyze these time series however is mostly centered around statistical models like regression models.

So if it comes to analyzing time series, topological data analysis is not the first approach that comes to mind. One reason is that the topology of time series is very simple since it only consists of one connected component and time series can not produce any topologically interesting shapes. Nevertheless we can circumvent this problem by transforming the time series into a structure that has the same dynamics but also carries a lot of topological information. In this work these structures will be graphs and even tho graph theory is a big mathematical field we will not use it directly to analyze them but we will rather use topology. Topology enables us to study the shape of these graphs and give a summary, a so called persistence diagram, about the topological features present in the graph. The goal is to use this summary to draw inference about the dynamics that determine the time series.

In this way we do not have to worry about the distribution that determines our dynamics in the beginning of the analysis. Rather do we get a toplogical description of the dynamics as an output from the analysis. Furthermore the analysis is self contained in that we can take any time series as an input and receive an output without making any a priori assumtions to the dynamical system itself which seems like a very beneficial property.

We will start with the mathematical theory needed to implement this method which consists of graph theory and topology.

Chapter 2

Graphs and Topology

2.1 Graph Theory

In this work graphs are the structures which we analyze with the help of Topology. These Graphs will be constructed from a time series and we will see that there are many different possibilities to do the construction, in particular when defining the edges.

Definition 1. *A Graph is a pair $G = (V, E)$ of sets satisfying $E \subseteq V^2$. Elements of V are called vertices and elements of E are called edges. The number of vertices of a Graph is its order. Two vertices v, w are adjacent if (v, w) is an edge of E.*

Definition 2. *A path is a non-empty graph $P = (V, E)$ of the form*

$$V = \{x_0, x_1, ..., x_k\}, E = \{(x_0, x_1), (x_1, x_2), ..., (x_{k-1}, x_k)\},$$

where the x_i are all distinct.

The vertices x_0 and x_k are *linked* by P and are called its *ends*; the vertices $x_1, ..., x_{k-1}$ are the *inner* vertices of P. The number of edges of a path is called its *length*.

Now that we know what a graph is lets look at some variations of graphs which will later be used in the analysis:

If all the vertices of G are pairwise adjacent, G is called a *complete graph* and denoted K^n.

A graph G is called *directed* if for $v, w \in V : (v, w) \neq (w, v)$ where $(v, w), (w, v) \in E$ are directed edges. We say that the edge (v, w) points from the vertex v to the vertex w.

G is called *weighted* if each edge has a numerical weight assigned to it, so if there exists a weight function $w\colon E \to \mathbb{R}$. If G is *weighted* the length of a path in G is the sum of the weights assigned to the edges of this path.

A Graph of order n can be stored in an adjacency matrix $M \in \mathrm{Mat}(n, n, \mathbb{R})$. If the vertices are ordered, i.e. $(v_1, ..., v_n)$, the matrix entry $m_{i,j}$ is $w((v_i, v_j))$ if (v_i, v_j) is an edge in E and 0 otherwise.

The simplicial complexes defined below can be seen as weighted undirected graphs embedded in euclidian space. However, the graphs looked at in this work usually do not have a natural embedding in euclidian space. So there is need for a more general notion of distance for graphs.

Definition 3. *The geodesic distance between two vertices $v, w \in V$ in a graph $G = (V, E)$ is defined to be the length of the shortest path connecting them.*

2.2 Simplicial Complexes

To analyze graphs we view them as simplicial complexes on which we can perform algebraic calculations. Every graph G discussed above defines an (abstract) simplicial complex K consisting of 0-simplices which are the vertices and 1-simplices which are the edges. Furthermore, every complete subgraph $K^p \subseteq G$ is an p-simplex in K.

Definition 4. *Let V be an euclidian vector space and $v_0, ..., v_n \subset V$ affinely independent. Define the by $(v_0, ..., v_n)$ spanned $n-simplex$*

$$\Delta(v_1, ..., v_n) = \{\sum_{i=0}^{n} t_i v_i \colon 0 \leq t_i \leq 1, \sum t_i = 1\} \subseteq V$$

where the v_i are called its edges and n its dimension.

$\Delta(v_1, ..., v_n)$ is the convex closure of $(v_1, ..., v_n)$ which is the smallest convex set that contains $(v_1, ..., v_n)$.

A *geometric simplicial complex* K is a set of affine simplices, such that for all simplices all its faces are also contained in K and for two simplices $\sigma, \tau \in K$ the intersection is either empty or a face of σ and τ.

Next we define abstract simplicial complexes which will represent our graphs as described above:

Definition 5. *An abstract simplicial complex K is a set of finite sets, such that for all $A \in K$ any subset $B \subseteq A$ is also contained in K:*

- $A \in K$ *and* $B \subseteq A \Rightarrow B \in K$
- $A \in K$ *is called a simplex*
- $dim(A) := |A| - 1$
- $B \subseteq A$ *is a face of* A

Geometric simplicial complexes and abstract ones are directly connected. Let K be a geometric simplicial complex and V the set of vertices of K. Then $\tilde{K} = \{\{a_0, ..., a_i\} \subseteq V | \Delta(a_0, ..., a_i) \in K\}$ is called the vertex-schema of K. $\tilde{K}$ is an abstract simplicial complex.

It is easy to shown that every abstract simplicial complex $\tilde{K}$ is isomorphic to the vertex-schema of a geometric simplicial complex and two geometric simplicial complexes are affine homeomorphic if and only if their vertex-schema are isomorphic.

Definition 6. *The mesh size of a simplicial complex K is defined to be*

$$mesh(K) := sup\{diam(\sigma) | \sigma \in K\},$$
$$diam(\sigma) := sup\{d(x, y) | x, y \in \sigma\}$$

where $d(x, y)$ is the distance between x and y.

2.3 Homology

Homological Algebra makes it possible to assign certain commutative groups, so called homology groups, to simplicial complexes. These groups are invariants of simplicial complexes.

To define these groups it is necessary to define an *orientation* on simplices, maps from the set of oriented simplices and the so called boundary operator:

An *ordering* of a $p - simplex$ $\sigma = \{v_0, \ldots, v_p\}$ is a bijective map

$$f \colon \{0, \ldots, p\} \to \sigma,$$

which is an ordering of the vertices. We can define an equivalence relation on the orderings of a simplex σ which identifies orderings with each other that differ by an even permutation, elements of the alternating group A_{p+1}:

$$f \sim g : \Longleftrightarrow \exists \pi \in A_{p+1} : g = f \circ \pi$$

Definition 7. *An orientation of a simplex σ is an equivalence class of orderings on σ.*

A *simplex* with an *orientation* is called an *oriented simplex* and an *oriented simplex* induces the orientation on all its faces. Any simplex has exactly two different orientations. For every simplicial complex K, $\hat{K_p}$ defines the set of oriented $p-simplices$ in K.

Now we can define maps from oriented p-simplices into $\mathbb{Z}$ which will enable us to define homology groups.

Definition 8. *A p-chain on K is a map $c\colon \hat{K_p} \to \mathbb{Z}$ such that*

1. *c vanishes only on finitely many simplices*
2. *$c(\hat{\sigma}) = c(\hat{\sigma}^{op})$, when op is the opposite orientation.*
3. *The set of all p-chains is called $C_p(K)$. It is a free commutative group. Furthermore we define $C_p(K) = \{0\}$ for $p < 0$ or $p > dim(K)$*
4. *For $\hat{\sigma} \in \hat{K}_p, p > 0$ define its elementary p-chain as*

$$\begin{aligned} \delta_{\hat{\sigma}}\colon \hat{K}_p &\to \mathbb{Z} \\ \hat{\sigma} &\mapsto 1 \\ \hat{\sigma}^{op} &\mapsto -1 \\ \hat{\tau} \notin \{\hat{\sigma}, \hat{\sigma}^{op}\} &\mapsto 0 \end{aligned}$$

If one chooses an orientation for all $\sigma \in K_p$, the set $\{\delta_{\hat{\sigma}} | \sigma \in K_p\}$ is a basis of $C_p(K)$. Between these chain groups we define a homomorphism called the *boundary operator*.

Definition 9. *The boundary operator is defined as follows on the basis:*

$$\begin{aligned} \partial_p\colon C_p(K) &\to C_{p-1}(K) \\ \delta_{[v_0,\ldots,v_p]} &\mapsto \sum_{i=0}^{p} (-1)^i \delta_{[v_0,\ldots,\hat{v}_i,\ldots,v_p]} \end{aligned}$$

where the component with the hat is deleted.

So for any simplicial complex of dimension d we get a sequence of homomorphisms called a *chain complex*.

$$\{0\} \xrightarrow{\partial_{d+1}} C_d(K) \xrightarrow{\partial_d} C_{d-1}(K) \xrightarrow{\partial_{d-1}} \ldots \xrightarrow{\partial_1} C_0(K) \xrightarrow{\partial_0} \{0\}$$

while $\partial_i \partial_{i-1} = 0$ for all $1 \leq i \leq d+1$.

Using the chain groups we can finally define homology groups:

Definition 10. *Let K be a simplicial complex, and let $C_p(K)$ be its chain groups.*

1. *The set $Z_p(K) := ker(\partial_p \colon C_p(K) \to C_{p-1}(K))$ is called p-cycles.*
2. *The set $B_p(K) := im(\partial_{p+1} \colon C_{p+1}(K) \to C_p(K))$ is called p-boundaries.*
3. *The commutative group $H_p(K) := Z_p(K)/B_p(K)$ is called the p-th homology group of K.*

These groups carry topological information about the simplicial complex: The equivalence classes in $H_p(K)$ are exactly the $(p+1)$-dimensional holes in the simplicial complex. The elements in $H_0(K)$ are the connected components of the simplicial complex K. These homology groups can easily be calculated using software like Sagemath.

2.3.1 Persistent Homology

After we defined general homology groups, the goal now is to define persistent homology groups which carry even more information about the topological space we try to approximate. They are able to not only find the holes in a simplicial complex but in a way they also tell us how big those holes are. To do this we first need to understand simplicial maps and how they induce maps on homology groups.

Let K and L be simplicial complexes. A simplicial map $f \colon K \to L$ is a map $f \colon V(K) \to V(L)$ from the corresponding sets of vertices such that for $\sigma \in K$ we have $f(\sigma) \in L$. Simplicial maps $f \colon K \to L$ induce chain maps $f_C \colon C_p(K) \to C_p(L)$, which can be defined on the basis:

$$f_C([v_0, ..., v_p]) = \begin{cases} [f(v_0), ..., f(v_p)], & \text{if } f(v_i) \neq f(v_j) \forall i \neq j; \\ 0, & \text{otherwise} \end{cases}$$

In this way we get a homomorphism of groups $C_p(K) \to C_p(L)$. The map f_C commutes with the boundary operator ∂_p.

Now we can define homomorphisms between homology groups which are induced by simplicial maps:

Definition 11. *Let K and L be simplicial complexes and $f\colon K \to L$ a simplicial map. The map f induces a homomorphism of groups*

$$f_H\colon H_p(K) \to H_p(L) \text{ with } f_H([c]) = [f_C(c)], \text{for } c \in C_p(K),$$

the identity $id\colon K \to K$ induces the identity on homology groups

$$id_H\colon H_p(K) \to H_p(K),$$

and if M is another simplicial complex and $g\colon L \to M$ another simplicial map, we have $(g \circ f)_H = g_H \circ f_H$.

Now lets consider not a single simplicial map but a finite sequence of simplicial maps K:

$$K\colon K_0 \xrightarrow{f_0} K_1 \xrightarrow{f_1} ... \xrightarrow{f_{N-1}} K_N$$

We denote the composition of these maps by $f^{i,j} = f_{j-1} \circ ... \circ f_i\colon K_i \to Kj$ for $i < j$. We have $f^{j,k} \circ f^{i,j} = f^{i,k}$.

Definition 12. *Let K be a finite sequence of simplicial maps. The p-th persistent homology group of K is defined as*

$$H_p^{i,j}(K) := im(f_H^{i,j}\colon H_p(K_i) \to H_p(K_j)) \subseteq H_p(K_j) =: H_p^j(K).$$

We write $H_p(K)\colon H_p(K_0) \xrightarrow{f_{0H}} H_p(K_1) \xrightarrow{f_{1H}} ... \xrightarrow{f_{(N-1)H}} H_p(K_N)$ for the corresponding sequence of homology groups.

2.3.2 Persistence Diagrams

There are many variations to homology groups. One which is particularly useful in our case is homology with coefficients. Here we define the $p-chains$ to be mapped to an arbitrary commutative group instead of $\mathbb{Z}$. If we consider homology with coefficients in a field $\mathbb{K}$ we achieve some simplifications:

1. $H_k(\circ, \mathbb{K})$ are $\mathbb{K}$ vector fields.
2. The induced maps $f_H\colon H_k(\circ, \mathbb{K}) \to H_k(\circ, \mathbb{K})$, are homomorphisms of the vector field $\mathbb{K}$.

So a finite sequence of simplicial maps induces a sequence of homomorphisms of finitely dimensional vector fields. This allows us to apply a theorem of representation theory, Gabriels Theorem.

Definition 13. *Let $N \in \mathbb{N}_0$. For $0 \leq a < b \leq N$*

$$I_{a,b} \colon 0 \to ... \to 0 \to \mathbb{K} \xrightarrow{1} ... \xrightarrow{1} \mathbb{K} \to 0 \to ... \to 0$$

is an elementary sequence of intervals of length N+1.

Theorem 1. *Let*

$$V_0 \xrightarrow{f_0} V_1 \xrightarrow{f_1} ... \xrightarrow{f_{N-1}} V_N$$

be a sequence of homomorphisms in the finitely dimensional vector field $\mathbb{K}$. Then there exist $(a_i, b_i) \in \{0, ..., N\}^2, 1 \leq i \leq r$ with

$$V \cong \bigoplus_{1 \leq i \leq r} I_{a_i, b_i} \tag{2.1}$$

and $\{(a_i, b_i) | 1 \leq i \leq r\}$ is uniquely determined.

To proof this statement Gabriels Theorem comes into play. It says that a representation of a quiver, which is a directed graph, can be decomposed into finitely many indecomposable representations if and only if the quiver is one of the ADE Dynkin diagrams. Now a sequence of vector field homomorphisms as defined above is a finite representation of the A_{n+1} quiver. Therefore Gabriels Theorem applies and the sequence is isomorphic to a finite sum of elementary sequences of intervals as described in equation 2.1.

This shows that any sequence of homology groups with coefficients in some field $\mathbb{K}$ is isomorphic to a finite sum of elemantary sequences of intervals:

$$H_p(K, \mathbb{K}) \cong \bigoplus_{0 \leq a < b \leq N} I_{a,b}^{y^{a,b}},$$

where $y^{a,b}$ denotes the multiplicity of the sequence $I_{a,b}$ in the decomposition. We can plot these $y^{a,b}$ in a so called Persistence Diagram, plotting a, which is called the birth of the homology class, on the x-axis and b, which is called the death of the homology class, on the y-axis. Let $x = (a, b)$ be a point in a persistence diagram P. The persistence of x is defined as $pers(x) = b - a$.

We can also define a distance between two persistence diagrams. To do this we first define the distance between two intervals.

Definition 14. *Let $I_1, I_2 \subset \mathbb{R}$ two intervals with boundaries $a_1 \leq b_1, a_2 \leq b_2$. The distance between them is defined as*

$$\Delta(I_1, I_2) := max\{|a_2 - a_1|, |b_2 - b_1|\}$$

For some interval $I \subset \mathbb{R}$ with boundary $a \leq b$

$$\lambda(I) := \frac{b-a}{2}$$

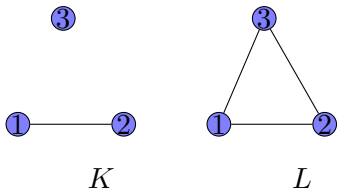
defines the distance to the nearest 1-point-interval $[c, c]$.

A Persistence Diagram with n points is a set of intervals $P = \{(a_i, b_i) \in \mathbb{R}^2 | 1 \leq i \leq n\}$. So we can define a distance between two persistence diagrams as follows:

Definition 15. *Let L and P be two persistence diagrams. The bottleneck distance between them is defined as*

$d_b \colon BC \times BC \to \mathbb{R}^{\geq 0}, (I, J) \mapsto d_b(I, J)$,
$d_b(I, J) := \inf\limits_{i : A' \subseteq A \hookrightarrow B \ inj.} \max\{ \sup\limits_{\alpha \in A'} \{\Delta(I_\alpha, J_{i(\alpha)})\}, \sup\limits_{\alpha \in A \setminus A'} \{\lambda(I_\alpha\}, \sup\limits_{\beta \in B \setminus i(A'} \{\lambda(J_\beta\}\}$.

The bottleneck distance is the shortest distance d for which there exists a perfect matching between the points of the persistence diagrams. If we have two diagrams with differently many points the remaining points are matched to the nearest 1-point-interval.



2.3.3 An Example

Lets consider two different simplicial complexes K and L. At first we will calculate their homology individually and then we define a simplicial map between them to calculate persistent homology.

Their corresponding abstract simplicial complexes are $\hat{K} = \{\{1\},\{2\},\{3\},\{1,2\}\}$ and $\hat{L} = \{\{1\},\{2\},\{3\},\{1,2\},\{2,3\},\{3,1\}\}$. Furthermore we have

$$\hat{K}_0 = \{\{1\},\{2\},\{3\}\} \qquad \hat{L}_0 = \{\{1\},\{2\},\{3\}\}$$
$$\hat{K}_1 = \{\{1,2\}\} \qquad \hat{L}_1 = \{\{1,2\},\{2,3\},\{3,1\}\}$$

which give the chain groups

$$C_0(K) = \{n_1\delta_{[1]} + n_2\delta_{[2]} + n_3\delta_{[3]} | n_1, n_2, n_3 \in \mathbb{Z}\}$$
$$C_1(K) = \{n_4\delta_{[1,2]} | n_4 \in \mathbb{Z}\}$$
$$C_0(L) = \{n_1\delta_{[1]} + n_2\delta_{[2]} + n_3\delta_{[3]} | n_1, n_2, n_3 \in \mathbb{Z}\}$$
$$C_1(L) = \{n_4\delta_{[1,2]} + n_5\delta_{[2,3]} + n_6\delta_{[3,1]} | n_4, n_5, n_6 \in \mathbb{Z}\}.$$

For such simple simplicial complexes the following calculations can be done by hand. However, since the goal is to give a practical approach to this topic I will only show how one can calculate the chain complex and homology groups with the help of Sagemath. To define a simplicial complex in Sagemath we only need to define the highest dimensional simplices and everyone of their faces will automatically be included. See Figure 2.1 for the results.

We see that 0-th homology group of K is isomorphic to $\mathbb{Z}\times\mathbb{Z}$ which means that we have two connected components and the first homology group is $\{0\}$ so there is no two dimensional hole in K. On the other hand, the 0-th as well as the first homology group of L is $\mathbb{Z}$, so we have one connected component and one two dimensional hole.

Note that $H_0(K)$ counts the connected components of K, $H_1(K)$ the two dimensional holes, $H_2(K)$ the three dimensional holes and so on.

```
[3]: K=SimplicialComplex([[1,2],[3]])
     C=K.chain_complex()

[4]: print(ascii_art(C))

                [-1]
                [ 1]
                [ 0]
     0 <-- C_0 <----- C_1 <-- 0

[5]: C.homology()

[5]: {0: Z x Z, 1: 0}

[6]: L=SimplicialComplex([[1,2],[2,3],[3,1]])
     C=L.chain_complex()

[7]: print(ascii_art(C))

                [-1 -1  0]
                [ 1  0 -1]
                [ 0  1  1]
     0 <-- C_0 <----------- C_1 <-- 0

[8]: C.homology()

[8]: {0: Z, 1: Z}
```

Figure 2.1: Calculations with Sagemath

Now lets define a simplicial map between the two simplicial complexes K and L to calculate persistent homology. It is obvious that $K \subseteq L$, so we just define the simplicial map $f\colon K \to L$ as the embedding of K into L:

$$\begin{aligned}\{1\} &\mapsto \{1\}\\ \{2\} &\mapsto \{2\}\\ \{3\} &\mapsto \{3\}\end{aligned}$$

which induces the homomorphisms on the homology groups

$$\begin{aligned} f_{H_0}\colon H_0(K) &\to H_0(L) & f_{H_1}\colon H_1(K) &\to H_1(L)\\ n_1[[1]] + n_2[[3]] &\mapsto n_1[[1]] & [[0]] &\mapsto [[0]]\end{aligned}$$

These homomorphisms give us the persistent homology groups

$$\begin{aligned} H_0^{K,L} &= \mathbb{Z}[[1]] \cong \mathbb{Z}\\ H_1^{K,L} &= \{0\}\end{aligned}$$

and we can see that the connected component $y_1 = [[3]]$ has persistence $[K, L)$ which means that it is born in K and dies before L, $y_2 = [[1]]$ has persistence $[K, \infty)$ and the hole $y_3 = [[1, 2] + [2, 3] + [3, 1]]$ has persistence $[L, \infty)$.

2.3.4 Different Complexes

In contrast to the example above we usually do not have the simplicial complex given. Instead we want to construct one from data which is usually point cloud data or in this case some time series. There exist many different complexes one can use, we now discuss the Vietoris-Rips Complex and the Clique Complex.

Vietoris Rips Complex

Let $S \subset X$ be a finite subset of a metric space X, for example a point cloud in euclidian space. The Vietoris Rips Complex is defined to be

$$VR_r = \{\sigma \subseteq S | diam(\sigma) < 2r\}$$

The Vietoris Rips Complex is dependent on the value of r. In particular for two values $r \leq r'$ we have $VR_r \subseteq VR_{r'}$, so we can define a simplicial map between them as the embedding $VR_r \hookrightarrow VR_{r'}$. Thus for $r_1 < ... < r_k$ we get a sequence of embeddings (simplicial maps)

$$VR_1 \hookrightarrow VR_2 \hookrightarrow ... \hookrightarrow VR_k$$

for which we can calculate persistent homology. Such a sequence is called a filtration.

Clique Complex

Let $G = (V, E)$ be a Graph. The Clique Complex of G is defined to be

$$Cl(G) = \{\sigma \subset V | \forall s, s' \in \sigma : (s, s') \in E\}$$

So every subgraph $S \subseteq G$, which is a complete graph, is a simplex in $Cl(G)$. If G is a directed graph, $\sigma = (v_0, ..., v_n)$ is a simplex in $Cl(G)$ if $(v_i, v_j) \in E$ for all $i < j$.

We can define a filtration on this complex by restricting which vertices of V are adjacent. Define $E_r := \{(s, s') | s, s' \in V \text{ and } d_g((s, s')) \leq r\}$ and the corresponding graphs $G_r = (V, E_r)$ where d_g is the geodesic distance. Note that these graphs depend on G, in particular on the edges in E. Again, for two values $r \leq r'$ we have $Cl(G_r) \subseteq Cl(G_{r'})$ since $G_r \subseteq G_{r'}$.

Thus, if we have a graph given we can compute persistent homology using the Clique Complex. The graph can be directed and or weighted.

Chapter 3

Method

Now that we know how to compute persistent homology of graphs, lets see how we actually construct the graph from some discrete time series. See [2] for an extensive review of different constructions.

3.1 Takens Embedding Theorem

Let T be a discrete time series. To get a data set which is more suited for our topological data analysis it is assumed that T is some measurement of a higher dimensional dynamical system S which we can reconstruct with a time delay embedding. We then perform the topological analysis on the reconstruction of the dynamical system S.

Lets assume T is a discrete scalar time series consisting of n values $t(1), t(2), ..., t(n) \in \mathbb{R}$. We can define a time delay embedding

$$\begin{aligned} s &: \mathbb{R} \to \mathbb{R}^d \\ t(k) &\mapsto s(t(k)) \\ s(t(k)) &= (t(k), t(k+c), t(k+2c), ..., t(k+dc)) \end{aligned}$$

for every $t(k)$ with $1 \leq k \leq n - dc$. The time delay c and dimension d of the embedding are parameters to be determined. We will discuss the two most common approaches to determine these parameters. As a result we get a dynamical system

$$S = (s(1), ..., s(n - dc - 1), s(n - dc)) \subset \mathbb{R}^d$$

of d-dimensional vectors in $\mathbb{R}^d$ and Takens Theorem [1] tells us that the dynamics in S are equivalent to those in T.

Theorem 2. *Takens Theorem. Let M_0 be a compact m-dimensional C^2-Manifold, $\phi\colon M_0 \to M$ a C^2-Diffeomorphism and $o\colon M_0 \to \mathbb{R} \in C^2(M_0, \mathbb{R})$. Then the function $\Phi_{\phi,o}\colon M_0 \to \mathbb{R}^{2m+1}$ with*

$$\Phi_{\phi,o}(x) := (o(x), o(\phi(x)), o(\phi^2(x), ..., o(\phi^{2m}(x)))$$

is in general an embedding.

This shows that starting from a time series we can construct a submanifold in $\mathbb{R}^{2m+1}$ which is diffeomorph to the underlying phase space. The condition that M_0 has to be a compact manifold was later weakened by Sauer, Yorke and Casdagli [6] and the dimension m they used is the $Boxcounting - Dimension$ of the attractor. The comment that Φ is 'in general' an embedding means that the set of (ϕ, o) for which $\Phi_{\phi,o}$ is not an embedding is a null set with regard to the product measure in $C^2(M_0, \mathbb{R}) \otimes Diff^2(M_0, M_0)$. This shows that the probability to have such functions (ϕ, o) is zero. Therefore it is reasonable to assume that the assumptions of this theorem hold in our case of scalar time series.

3.1.1 Determining the Parameters for Takens Embedding

There are several different ways to calculate the parameters for Takens Embedding. Even tho the time delay parameter can be arbitrary for Takens Embedding to hold, in praxis the right choice of time delay is important.

The most common approach and also the approach I use in this work is to calculate the dimension with a false nearest neighbor algorithm [4] and to use a mutual information function to determine a suitable time delay, which was first introduced by [3].

The problem for small time delays c is that successive values of a time series can be strongly correlated and nearly identical. The mutual information function tries to address this problem and is an extension to the autocorrelation function which is only capable to detect linear correlations. With the help of Shannon Entropy the mutual information function is also capable of finding non linear correlations in our data such that the components of our embedded vectors are as independent as possible.

To find a proper embedding dimension the idea is to find false neighbors of points which are only neighbors because the embedding dimension is too small. The algorithm increases the embedding dimension until distances between neighboring points do not change significantly anymore.

3.2 Permutation Sequences

After the embedding of our time series is calculated we have a dynamical system of d-dimensional vectors $s(1), \ldots, s(n - dc) \in \mathbb{R}^d$. It is possible to just calculate the persistent homology of the Vietoris Rips Complex of this point cloud. However, the information about how the system evolves over time would be lost. Furthermore it can be practical to further approximate the space which we analyze which is why we transform the point cloud into a permutation sequence.

To do this we assign a permutation $\pi \in S_d$ to each vector of the point cloud $s(1), \ldots, s(n - dc)$ such that for each vector

$$s(k) = (t(k), t(k + c), t(k + 2c), ..., t(k + dc)),$$

π satisfies $t(\pi(k)) \leq t(\pi(k+c)) \leq \cdots \leq t(\pi(k+dc))$. This gives a sequence of permutations $\pi_1, \ldots, \pi_{n-dc}$ with $\pi_i \in S_d$. From this permutation sequence we can construct a graph which we can analyze with the tools of persistent homology.

3.3 Ordinal Partition Graphs

Given a permutation sequence we now construct an Ordinal Partition Graph $G = (V, E)$.

The vertices V in G are simply the distinct permutations which appear in the permutation sequence and there is an edge between two vertices if and only if one permutation follows directly after the other. By Definition, the graph is directed and one can define weights on the edges by counting how often the edges are encountered.

To sum up, we start with a discrete time series T. We reconstruct the phase space of this time series and get an d-dimensional embedding. From this embedding we construct a permutation sequence which defines an ordinal partition graph which we eventually analyze with the help of persistent homology. The following chapter explains how this method can be implemented with Python.

Chapter 4

Implementation in Python

In this work I used the Giotto-tda [7] package. It has many steps of the method above already implemented like Takens Embedding for example. Lets start right there with a discrete time series T.

With the class *SingleTakensEmbedding* we can transform a scalar time series into a point cloud:

```
from gtda.time_series import SingleTakensEmbedding

ts = [some time series]
STE = SingleTakensEmbedding(parameters_type='search', dimension=10,
    time_delay=100)
ts_embedded = STE.fit_transform(ts)
```

If the parameter *parameters_type* is set to 'search' this transformer will calculate the optimal embedding parameters with a false nearest neighbor algorithm and the mutual information function. The parameters *dimension* and *time_delay* set the maximum values for the corresponding embedding parameters.

One can also calculate the optimal embedding parameters directly:

```
from gtda.time_series import takens_embedding_optimal_parameters

paras = takens_embedding_optimal_parameters(ts, 10, 100)
time_delay = paras[0]
dimension = paras[1]
```

Now we want to construct a permutation sequence from the point cloud. The *numpy* functions *argsort* and *unique* do exactly what we need:

```
import numpy as np

ps = np.argsort(ts_embedded)
u = np.unique(ps, axis=0, return_inverse=True)
```

And we can now construct an Ordinal Partition Graph from the permutation sequence. The following code computes a simple graph, i.e it is not directed and not weighted.

```
from scipy.sparse import csr_matrix

n = len(u[0])
row = []
col = []
data = []

adj = [[0 for permutation in u[0]]for permutation in u[0]]

for x in range(0,len(u[1])-1):
    a = u[1][x]
    b = u[1][x+1]

    if a != b and adj[a][b] == 0:
        adj[a][b] += 1
        row.append(a)
        col.append(b)
        data.append(1)

transition_graph = csr_matrix((data, (row,col)), shape=(n,n))
```

The transition graph is stored in a csr-matrix from scipy. Finally we compute the persistent homology. To do this we first need to compute a distance matrix from the graph:

```
from gtda.graphs import GraphGeodesicDistance
from gtda.homology import VietorisRipsPersistence

dm = GraphGeodesicDistance().fit_transform([transition_graph])
vr = VietorisRipsPersistence(metric='precomputed',
          homology_dimensions = (0,1,2))
hom = vr.fit_transform(dm)
```

Note that the complex we used is the Vietoris Rips Complex. Since we compute the homology of an undirected graph this produces the same output as if we used the Clique Complex but with a shorter runtime.

4.1 Directed and Weighted Graphs

If however we do not want to loose the information about how the system evolves over time we need to consider directed graphs with weighted edges. There are only slight changes in the code necessary to do this. To have weighted edges we need to define some weight function on the edges. The approach I used is to count how often each edge is encountered and to assign smaller weights to often encountered edges and bigger weights to less often encountered edges.

Lets say the permutation sequence runs through an edge e_1 k-times, while the edge e_{max} with the most encounters is run through n-times, $n \geq k$. In the graph the edge e_1 will have the weight $w(e_1) = n + 1 - k$ and the edge e_{max} will have the weight $w(e_{max}) = n + 1 - n = 1$ assigned to it.

```
adj = [[0 for permutation in u[0]]for permutation in u[0]]

for x in range(0,len(u[1])-1):
    a = u[1][x]
    b = u[1][x+1]

    if a != b:
        adj[a][b] += 1

max_entry = 0

for i in adj:
    for j in i:
        if j >= max_entry:
            max_entry = j
```

Here we first calculate an adjacency matrix for the permutation sequence and then we calculate how often the maximum edge is encountered. After that we weigh the edges in the graph as described above:

```
n = len(u[0])
row = []
col = []
data = []

adj = [[0 for permutation in u[0]]for permutation in u[0]]

for x in range(0,len(u[1])-1):
    a = u[1][x]
    b = u[1][x+1]

    if a != b:
        if adj[a][b] == 0:
            adj[a][b] += 1
            row.append(a)
            col.append(b)
            data.append(max_entry)

        else:
            row.append(a)
            col.append(b)
            data.append(-1)

transition_graph = csr_matrix((data, (row,col)), shape=(n,n))
```

To have the persistent homology computed for the Clique Complex we just need to specify in the transformer GraphGeodesicDistance that the graph we work with is a directed one. Then we use the class FlagserPersistence which computes the Clique Complex of our graph.

```
from gtda.homology import FlagserPersistence

ggd = GraphGeodesicDistance(directed=True)
dm = ggd.fit_transform([transition_graph])

fp = FlagserPersistence(homology_dimensions = (0,1,2), n_jobs=-1)
hom = fp.fit_transform(dm)
```

A jupyter notebook with outputs from the code above can be found in Appendix A.

Chapter 5

Applications

In this chapter we will apply the method to some datasets and look for possibilities to work with the resulting persistent homology.

5.1 Periodic and Chaotic Time Series

Firas et al [5] used this method for dynamic state detection. They showed that given a time series with periodic and chaotic sections they can distinguish between them by the information given in their persistent homology.

The approach they used is not to compare the persistence diagrams directly but rather compute point summaries of a persistence diagram.

To be exact, they used a time delay embedding of their time series for which they computed the parameters using mutual information and false nearest neighbor. Then they defined a permutation sequence for the embedding and constructed an undirected and unweighted ordinal partition graph from it. The graph was analyzed with persistent homology and the following point summaries were extracted from the persistence diagram:

Definition 16. *Let P be a persistence diagram of a graph $G = (V, E)$.*

- *The maximum persistence of P is defined as*

$$maxpers(P) = \max_{x \in P} pers(x)$$

- *The Periodicity Score of P is defined as*

$$S(P) = 1 - \frac{maxpers(P)}{C_n},$$

where C_n is the maximum persistence of a cycle graph with n vertices: $C_n = \lceil \frac{n}{3} \rceil - 1$.

- *The ratio of the number of homology classes to the graph order is defined as*

$$M(P) = \frac{|P|}{|V|}$$

- *The Normalized Persistent Entropy E' is defined as*

$$E'(P) = \frac{E(P)}{log_2(L(P))}$$

with $L(P) = \sum_{x \in P} pers(x)$ *and*

$$E(P) = -\sum_{x \in P} \frac{pers(x)}{L(P)} log_2(\frac{pers(x)}{L(P)})$$

These point summaries were compared and showed strong differences for periodic and chaotic time series as long as the signal to noise ratio was small enough.

However, by only considering undirected and unweighted graphs valuable information about the dynamical system is lost. Thus in the following I try to change the method by using directed and weighted graphs. How the weights and directions are defined is described in section 4.1.

The Rössler System

Lets have a look at the Rössler System to compare the approach with different graphs. At first we just compare the persistence diagrams to get an understanding on how changing the graph changes the persistent homology. The time series we use is a chaotic section of the Rössler System with parameters $a = 0.2, b = 0.2, c = 6$, it is plotted in Figure 5.1:

We notice that the more complex the graph is, the better we can differentiate between homology classes. If we look at weighted graphs it becomes apparent from H_0 how often the different edges are visited during the permutation sequence. Furthermore the births and deaths of the homology classes differ due to the different weights assigned to the edges. Thus we seem to get a more complete picture of the dynamics in our time series.

If however we want to apply the point summaries to the persistence diagrams we see quite fast that they are not robust to changes that we

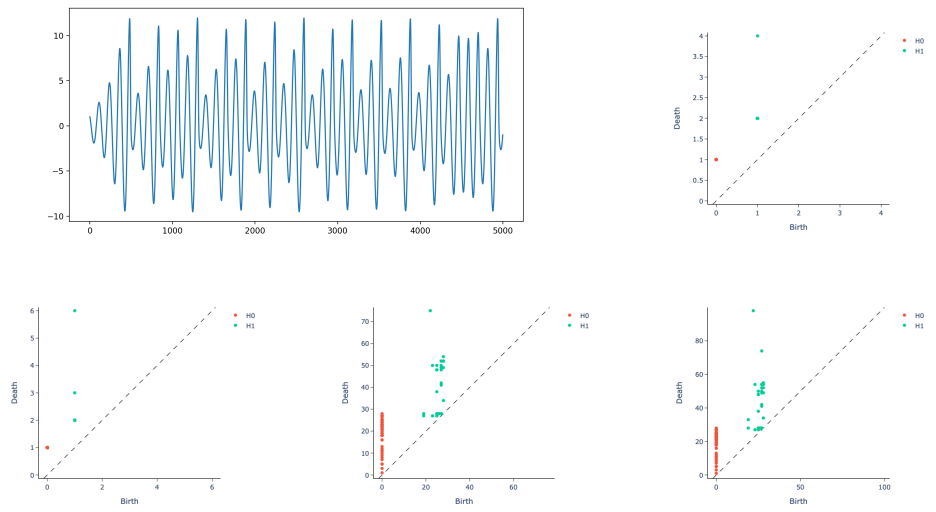


Figure 5.1: From top left to bottom right: chaotic section of the Rössler System, Persistence diagrams for a Simple Graph, Directed Graph, Weighted Graph and Directed plus Weighted Graph.

make to our graphs. The periodicity score for example compares the input graph with a cycle graph which is neighter directed nor weighted. So we should rather compare weighted graphs to a weighted cycle graph but it is not clear which weights one should assign to the edges of the cycle graph. Besides the Periodicity Score, the ratio of homology classes to graph order and the normalized persistent entropy still give reasonable results, however they do not seem to be comparable with the summaries for simple graphs. The normalized entropy takes values in the interval $[0, 1)$ and a value near 0 is indicative for periodic dynamics while a value near 1 is indicative of a chaotic dynamic. For directed and weighted graphs however its value for chaotic systems lies around 0.5 and for periodic systems around 0.2. So there is still an apparent difference but it is less significant. The ratio of homology classes to the graph order gives the best results for weighted and directed graphs but is not conclusive on its own. See Table 5.1 for some examples.

In conclusion, if one wants to take the approach with point summaries it is important to define special point summaries which match the graph one is using. The point summaries of Firas et al work best with a simple graph without weighted and directed edges.

	Simple Graph	DiGraph
Periodic Rössler	0, 0.08, 0	-1, 0.08, 0
Chaotic Rössler	0.66, 0.2, 0.82	0.5, 0.2, 0.65
Financial	0.94, 0.06, 0.83	0.94, 0.49, 0.97
	Weighted	Weighted DiGraph
Periodic Rössler	0, 0.08, 0	-1, 0.08, 0
Chaotic Rössler	-1, 0.25, 0.4	-3.3, 0.25, 0.32
Financial	0.69, 0.06, 0.53	-4.2, 1.35, 0.52

Table 5.1: Point summaries for different graphs: Periodicity Score, Ratio of Number of Homology Classes to Graph Order, Normalized Persistent Entropy.

5.2 Change Point Detection

Another Application is to try and find change points in a time series. To do that we split the time series into n pieces of the same size. Then we perform the analysis on each of these intervals and get n persistence diagrams. These persistence diagrams can be compared using the bottleneck distance. As a result we should see small bottleneck distances between intervals if the dynamics do not change much and larger bottleneck distances if the dynamics in the time series change.

To perform this kind of analysis we require datasets which are large enough to split them in many smaller parts such that these parts are still suited for the method. Furthermore, performing many calculations on many different intervals can be computationally very expensive, especially if we want to compute higher dimensional homology groups.

Nevertheless, lets see how this analysis might work on some time series: At first we need to consider which parameters we use for the embeddings. Consider some periodic time series like the cosine function. If we calculate persistent homology for two different embedding dimensions $d_1 << d_2$ the bottleneck distance between them can be arbitrarily large even tho they describe the same dynamic. So we should use the same embedding dimension for every interval. Therefore, to capture the dynamics in every interval correctly we have to calculate the optimal embedding parameters for every interval and then take the maximum of the embedding dimensions. The parameter for the time delay can be chosen arbitrarily for Takens Theorem to hold.

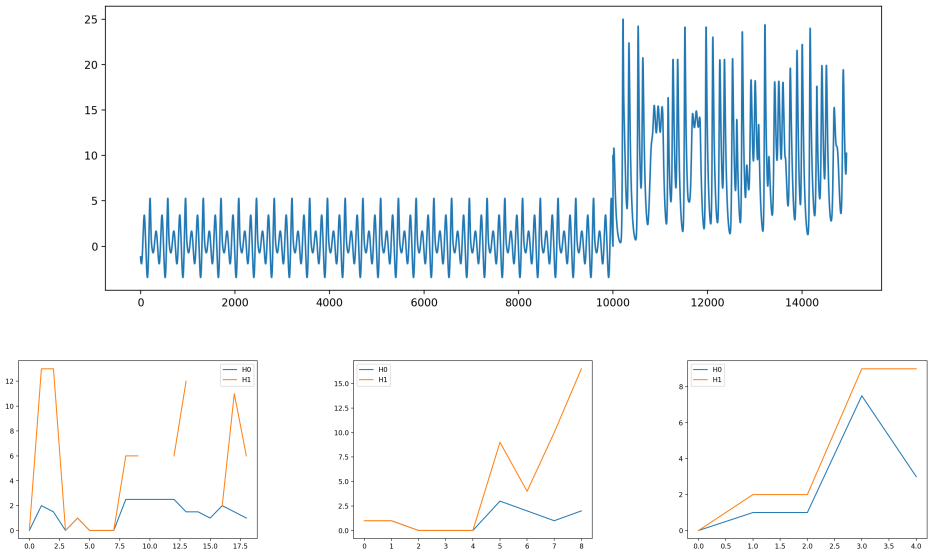


Figure 5.2: Top row: Time Series of Rössler System and Lorenz System. Bottom Row, left to right: Bottleneck Distances, intervals 750,1500,2500.

The time series used is a periodic section of the Rössler System followed by a chaotic section of the Lorenz system. So the change point we try to identify is located at $x = 10000$. The bottleneck distance is plotted for three differently sized intervals (750,1500,2500) and for the homology groups H_0 and H_1. We see that for the interval size 2500 there is a peak in the bottleneck distances which correspond to the change point in the time series. The smaller intervals however do not capture the change point since there is no peak at the transition. So choosing the right interval size is a crucial step which seems to depend on the duration of the different dynamics present in the time series. Since the time series of Figure 5.2 consists of only two different dynamics which occupy one third and two thirds of the dynamic a bigger interval is producing better results than smaller intervals. If we had many smaller dynamics, smaller intervals would be appropriate to detect change points. This assessment seems very difficult to do if we do not have any prior knowledge about the dynamics of the time series.

In Conclusion, the application of the method to detect change points seems to have potential. It is however difficult to put in to practice due to the computational complexity and the uncertainty about the size of the intervals.

5.3 Multivariate Time Series

The method can also be applied to multivariate time series as described in [8]. For a two dimensional time series Zhang et al simply took the cartesian product of the permutations $S_n \times S_n$. Lets say we have an embedded two dimensional time series $(x(t), y(t))$. The elements of the permutation sequence are tuples of permutations$(\pi_{x(t)}, \pi_{y(t)})$. Obviously this can be generalized to arbitrary dimensions.

However, using this produces many vertices in the ordinal partition graph very fast which increases the runtime for this method a lot. Since the degree k of the group S_k depends on the dimension parameter used in the time delay embedding $d = k$, for a n-dimensional time series the ordinal partition graph has up to $(k!)^n$ vertices, depending on the complexity of the time series.

Two combat this problem it is necessary to find other ways to approach multivariate time series. The basic idea stays the same however: We want to split the phase space into several chambers, so called Weyl chambers. While the cartesian product of symmetric groups is partitioning the space into very small chambers there exist many other Weyl chambers that can be used in this method and choosing the right ones may result in a partition of the phase space which still carries enough information about the dynamics but is computationally manageable.

This however assumes that the person analyzing the time series has enough knowledge to choose these Weyl chambers in a practical way.

Chapter 6

Conclusion

Analyzing time series with persistent homology seems like an approach with many opportunities. On the one hand we can change the transition graph to fit the application that we want to research. For example changing the weights assigned to the edges in a particular way can result in graphs which give particular information about the dynamics. Or we can define totally different graphs like the k-nearest-neighbor graph which may have certain advantages in some applications. On the other hand there are many possibilities to work with the persistence diagrams from these graphs. We can use point summaries like Firas et al to detect dynamic states or we may define different point summaries fitted to the graph in use to detect different characteristics of the dynamics. We can also use the bottleneck distances between persistence diagrams to try and calculate difference in the dynamics directly.

There are however many challenges that come with these possibilities like we can see with the application for change point detection. There may be some configuration to the method such that change points can be detected, without any prior knowledge however it may be very hard to determine these configurations and it can easily result in systematic errors for the method.

All in all it is however an interesting novel approach to time series analysis with many possibilities to apply topological data analysis and incorporate own ideas into the method.

The code for all of the different methods mentioned in this work can be found on my github page.

Appendix A

Jupyter Notebook

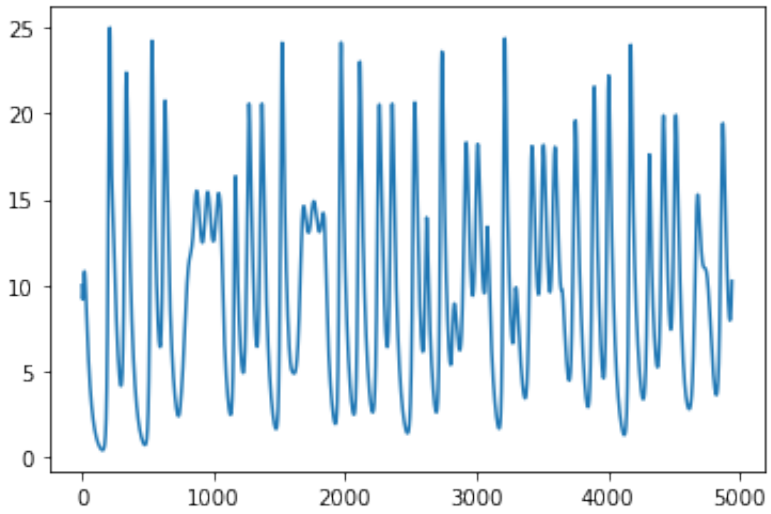
Persistent Homology of a Chaotic Section from the Lorenz System with a Directed and Weighted Graph

January 24, 2022

[13]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import networkx as nx
from scipy.sparse import csr_matrix
from time_series import Rossler, Lorenz, Cos
from gtda.time_series import SingleTakensEmbedding
from gtda.time_series import takens_embedding_optimal_parameters
from gtda.graphs import GraphGeodesicDistance
from gtda.homology import FlagserPersistence
from gtda.plotting import plot_point_cloud
```

[14]:
```
ts_y = Lorenz.image
```

[15]:
```
plt.plot(ts_y)
plt.show()
```

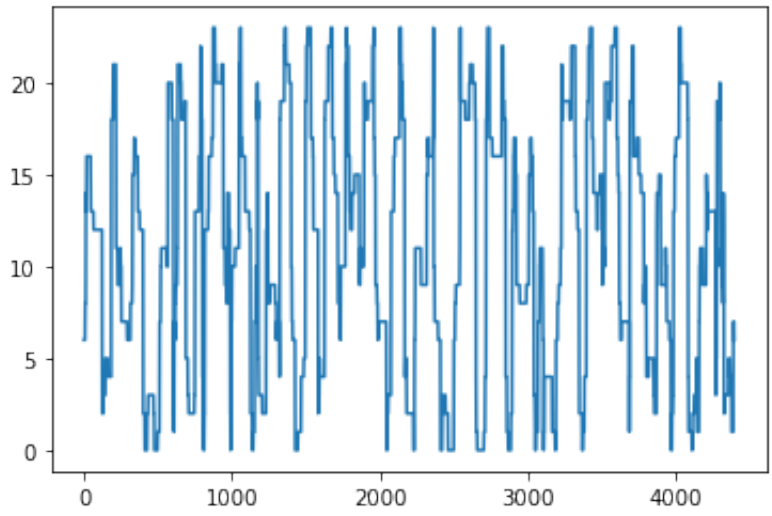


[26]:
```
par = takens_embedding_optimal_parameters(ts_y, 1000,10)
print(par)
```

```
(183, 4)
```

[18]:
```
STE = SingleTakensEmbedding(parameters_type='fixed',
                            time_delay=183,dimension=4)
ts_embedded = STE.fit_transform(ts_y)
```

[19]:
```
ps = np.argsort(ts_embedded)

u = np.unique(ps, axis=0, return_inverse=True)
y_axis = u[1]

plt.plot(y_axis)
plt.show()
```



From the Permutation Sequence create the Ordinal Partition Graph

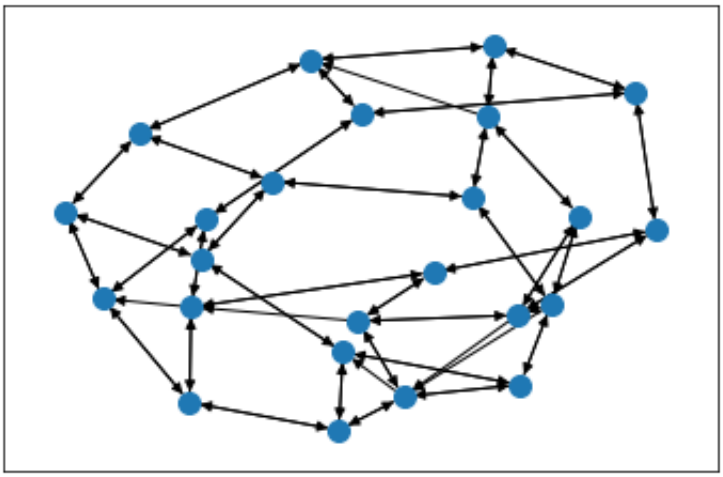
[20]:
```
adj = [[0 for permutation in u[0]]for permutation in u[0]]

for x in range(0,len(u[1])-1):
    a = u[1][x]
    b = u[1][x+1]
```

```
    if a != b:
        adj[a][b] += 1
```

[21]:
```
max_entry = 0

for i in adj:
    for j in i:
        if j >= max_entry:
            max_entry = j
```

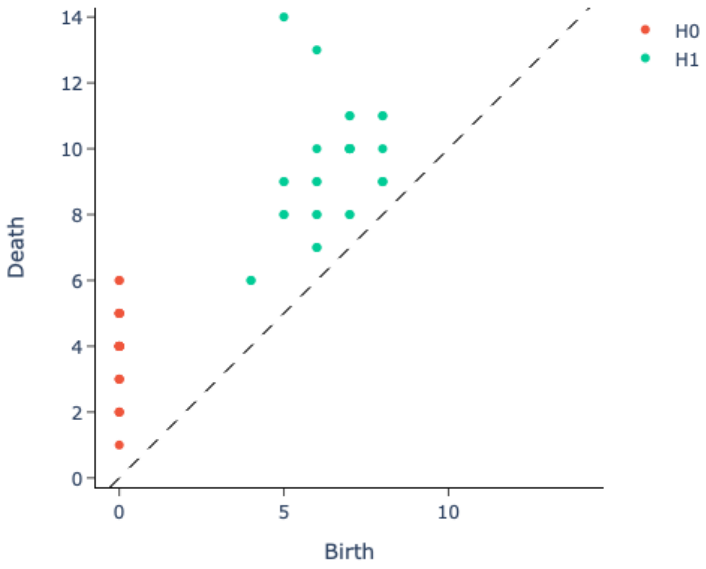
[22]:
```
n = len(u[0])
row = []
col = []
data = []

adj = [[0 for permutation in u[0]]for permutation in u[0]]

for x in range(0,len(u[1])-1):
    a = u[1][x]
    b = u[1][x+1]

    if a != b:
        if adj[a][b] == 0:
            adj[a][b] += 1
            row.append(a)
            col.append(b)
            data.append(max_entry)

        else:
            row.append(a)
            col.append(b)
            data.append(-1)

transition_graph = csr_matrix((data, (row,col)), shape=(n,n))
```

[23]:
```
tg = np.any(transition_graph)
g = nx.from_scipy_sparse_matrix(tg, create_using=nx.DiGraph)
nx.draw_networkx(g,node_size=100,with_labels=False)

plt.show()
```



[24]:
```
ggd = GraphGeodesicDistance(directed=True)
dm = ggd.fit_transform([transition_graph])
```

[25]:
```
fp = FlagserPersistence(homology_dimensions = (0,1), n_jobs=-1)
hom = fp.fit_transform(dm)
fp.plot(hom)
```



Bibliography

[1] Takens f. (1981) detecting strange attractors in turbulence. in: Rand d., young ls. (eds) dynamical systems and turbulence, warwick 1980. lecture notes in mathematics, vol 898. springer, berlin, heidelberg. pp.366-381.

[2] Reik V. Donner, Michael Small, Jonathan F. Donges, Norbert Marwan, Yong Zou, Ruoxi Xiang, and Jürgen Kurths. Recurrence-based time series analysis by means of complex network methods. *International Journal of Bifurcation and Chaos*, 21(04):1019–1046, Apr 2011.

[3] A.M. Fraser and H.L. Swinney. Independent coordinates for strange attractors from mutual information. 1986.

[4] Henry D.I. Abarbanel Matthew B. Kennel, Reggie Brown. Determining embedding dimension for phase-space reconstruction using a geometrical construction. 1992.

[5] Audun Myers, Elizabeth Munch, and Firas A. Khasawneh. Persistent homology of complex networks for dynamic state detection. *Physical Review E*, 100(2), Aug 2019.

[6] J.A. Yorke Sauer T. and M. Casdagli. Embedology. 1991.

[7] Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Wojciech Reise, Anibal Medina-Mardones, Alberto Dassatti, and Kathryn Hess. giotto-tda: A topological data analysis toolkit for machine learning and data exploration, 2021.

[8] Jiayang Zhang, Jie Zhou, Ming Tang, Heng Guo, Michael Small, and Yong Zou. Constructing ordinal partition transition networks from multivariate time series. *Scientific Reports*, 2017.