TOPOLOGICAL APPROACHES FOR QUANTIFYING THE SHAPE OF TIME SERIES DATA

By

Sarah Tymochko

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computational Mathematics, Science and Engineering – Doctor of Philosophy

2022

ABSTRACT

TOPOLOGICAL APPROACHES FOR QUANTIFYING THE SHAPE OF TIME SERIES DATA

By

Sarah Tymochko

Topological data analysis (TDA) is field that started only two decades ago and has already shown promise both in theory and in applications. The goal of TDA is to quantify the shape of data in a manner that is concise and robust using concepts from algebraic topology. Persistent homology, arguably the most popular tool from TDA, studies the shape of a filtered space by watching how its homology changes. The output of persistent homology is a persistence diagram, which encodes information about the changing homology.

Persistent homology has shown success in various application areas; one ever growing area of study in this field is time series analysis. Nonlinear time series analysis is a research field in and of itself that aims to capture structure in time series data, however, it lacks theoretically justified tools to analyze the resulting structure. Persistent homology comes with a solid theoretical framework, is robust to noise, and quantifies the same type of structure as appears in time series data. Thus combining tools from time series analysis and TDA provides a new approach to analyze and quantify behavior in time series data.

One field where time series are prevalent is dynamical systems, since a time series arises from a projection of a solution to a system. Specifically, given a time series, Takens' theorem can be leveraged to embed the time series as a point cloud in a higher dimensional space, where this point cloud is a sampling of the full state space. Then for each time series, persistent homology can be computed on the embedding. The result is a persistence diagram for each time series. The question then becomes how do we analyze this collection of persistence diagrams to learn something about the original time series data? Many people have developed methods to answer this question, through methods such as machine learning or statistics. This dissertation provides several new methods leveraging tools from both TDA and nonlinear time series analysis to study time varying data.

To my sister and my parents.

ACKNOWLEDGEMENTS

I know I would not have made it this far without the support of so many incredible people. I'm very sorry to anyone I've forgotten.

First and foremost my family. My sister, Hannah, has been my best friend and number one supporter my entire life. I could not have survived the highs and lows of the past five years without you, Han. My parents, Maureen and John, have always supported my decisions, no matter how crazy and unexpected. They have both sacrificed a lot for me to ensure I received a good education and I am so grateful to them both. Mom, your frequent visits to Michigan have been a lifesaver and I can't wait for our next trip to Throwback. Dad, you always encouraged me to pursue a PhD and I can't thank you enough for that. My aunt Patty provided a safe haven when I needed an escape from Michigan and makes the best chocolate chip cookies. And I can't forget my pandemic puppy, Bella, who helped keep me sane during the pandemic and the whole dissertation writing process.

I cannot imagine a better advisor than Liz Munch. Liz, thank you for your unwavering support the past five years. You are an incredible mathematician, mentor, teacher and human being, and I'm in awe of your ability to be all of that at once. You've been an inspiration to me from my first day in CMSE and I've learned so much from you over the past five years. I will miss our weekly chats and chanting "it's fine, everything is fine."

Also thank you to my (current and former) committee members, Jose Perea, Firas Khasawneh, Yuying Xie, and Matt Hirn for their feedback and support. I am also incredibly grateful for my undergraduate advisor, David Damiano, who first got me interested in research and encouraged me to apply to grad school. Tegan Emerson has been a fantastic mentor and collaborator, and I'm honored to have been able to intern with her. Thank you to Adam Alessio, Vanessa Robins, Giseon Heo, Tim Doster, Emilie Purvine and all the other people who have made a lasting impact on my career thus far. I feel so fortunate to have met so many fabulous people over the past five years.

I'm also grateful to have met so many amazing people through CMSE. Luke Stanek has been a source of endless support. Luke, you were my first friend in the department, bonding over battleship

and type racer (even though I always beat you in both), and surviving qual classes together. I can't thank you enough for putting up with my insanity. You have been there for me through so much and I don't know how I would have made it to this point without you. Nat Hawkins, thank you for always being available for advice or just a good vent session. Sarah McGuire, thanks for being my name twin and for all the walks with the pups. Nat, Luke, and Sarah (aka the Food Truck), I wouldn't have survived without our video game nights, lunches in the International center, and many many Costco or Hungry Howies pizzas. Also, thank you to Sarah Hawkins for tolerating our Food Truck gatherings and for being yet another name twin (there can never be too many Sarahs). Melinda McEwan, dinners with you, your family, and the pups helped keep me sane. I appreciate your constant willingness to help with anything and everything in the department. Also thank you to Melinda, Heather, Lisa, and the rest of the front office staff, I don't know how the department would function without you all. To the MunchLab members I've had the pleasure of working along side over the past five years, thank you for providing such a welcoming and supportive community. I will miss the many Dairy Store trips and Switch parties.

Despite being dispersed all over the country, the support of Lillie Reder, Alex McKenna, Cassie Naimie and Paige Severance has meant the world to me. Zoom chats with you all have been one of the best things to come out of the pandemic.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

Topological data analysis (TDA) is a collection of methods to quantify the shape of data. By leveraging tools from algebraic topology, TDA tools can capture shape such as connected components, loops and voids, and summarize that information in a format that is concise, robust, and comparable. Specifically, persistent homology encodes the structure of a filtered space in a persistence diagram.

One application area where persistent homology is particularly well suited is time series analysis. There are tools, specifically the time delay embedding or Takens' embedding [4], to embed time series data in $\mathbb{R}^d$ ($d \geq 2$) where underlying features of the original time series give different shapes in the embedding space. For example, periodic time series appear as a collection of points tracing out a circle in $\mathbb{R}^d$. Since persistent homology is designed to detect circular structures, it is a natural combination to quantify the shapes formed from these embeddings using persistence diagrams. This combination of persistent homology applied to time delay embeddings is well studied and has shown success in many applications [5–23].

This dissertation develops TDA tools for time series analysis in two ways: (1) by applying and modifying existing methods for applications and (2) by developing new methods to characterize behavior that previously has not been studied using persistent homology techniques. Under the first category, we combine existing methods from image processing and TDA to study time series satellite imagery from hurricanes to detect a daily cycle. Next, we develop a modification of a method of transforming persistence diagrams into feature vectors that can be used for machine learning tasks. We tested this method by classifying periodic or chaotic behavior in the Rössler dynamical system. Lastly, we attempt to avoid potential pitfalls of the time delay embedding, specifically the fact that longer time series generate larger embeddings and can become computationally prohibitive. We use a coarser embedding called ordinal partition networks, which embed a time series into a network, rather than a point cloud. Previous work has shown that using the topological structure of these networks provides a method of classification between periodic and chaotic time series. We

extend this work to incorporate additional features of the network into the topological analysis.

In the second category, we use a generalization of persistent homology called zigzag persistence to develop a one-step method of analyzing Hopf bifurcations in dynamical systems. This method bypasses the problem of analyzing a collection of persistence diagrams, resulting in only one persistence diagram that encodes information about when a bifurcation occurs. Further, we propose future work to improve this method by studying the zigzag persistent homology of ordinal partition networks to reduce computation time of our existing method.

This dissertation is structured in the following way: Chapter 2 provides the necessary background material from TDA [24, 25] as well as time series analysis [26], while Chapters 3–5 cover each of the four projects described. Note that a significant portion of the work in this dissertation has already been published in [27–29].

CHAPTER 2

BACKGROUND

In this section, we will cover background material from topological data analysis [24, 25] and time series analysis [26]. Additional background will be introduced in the relevant chapters, but tools such as homology, persistent homology, and the time delay embedding form the basis for all other methods utilized.

2.1 Homology

Homology is a standard tool in algebraic topology to study topological structure in different dimensions. In particular, given a space X, homology computes a group for dimensions $k = 0, 1, 2, \ldots$, denoted $H_k(X)$, that represents information about the structure in each dimension. In particular, dimension 0 studies connected components, dimension 1 studies loops, dimension 2 studies voids, and higher dimensions study the higher dimensional analogues.

We will first introduce a few other concepts in order to define homology, specifically simplicial homology, more formally. A simplicial complex $\mathcal{K}$ is a space built from different dimensional building blocks called simplices. These spaces can be viewed both geometrically and abstractly. Geometrically, an n-simplex σ is the convex hull of $n + 1$ affinely independent points, and a face of an n-simplex $\tau \leq \sigma$ is defined to be the convex hull of a nonempty subset of the vertices of σ. The simplicial complex must satisfy the following requirements: (1) the intersection of any two simplices in $\mathcal{K}$ is also a simplex in $\mathcal{K}$ and (2) all faces of a simplex in $\mathcal{K}$ are also simplices in $\mathcal{K}$. Abstractly, a p-simplex can be represented by the unordered set of $p + 1$ vertices it is built from. So a simplicial complex, $\mathcal{K}$, is a family of sets that is closed under taking subsets. That is, given a p-simplex, $\sigma \in \mathcal{K}$, then any simplex consisting of a subset of the vertices of size $0 < k \leq p$, called a k-dimensional face of σ, is also in $\mathcal{K}$.

For a given simplicial complex $\mathcal{K}$, let $\mathcal{K}_p$ be the set of all p-simplices, $p = 0, 1, 2, \ldots$. Then a

p-chain, c, is defined to be a finite[1] formal sum of p-simplices in $\mathcal{K}$,

$$c = \sum_{\sigma_i \in \mathcal{K}_p} a_i \sigma_i,$$

where coefficients $a_i \in \mathbb{Z}_2$. Note that other fields can be used for coefficients, but we will focus on the simplified case of $\mathbb{Z}_2$ as that is typically what is used for persistent homology. Since we can add and scale chains by a constant, the collection of p-chains, $C_p(\mathcal{K})$, called the chain group, forms a vector space. The boundary map between chain groups is defined as the linear transformation

$$\partial_p : C_p \to C_{p-1}$$

which maps a p-simplex to the sum of its $(p-1)$-dimensional faces. The chain complex is a sequence of chain groups connected by the corresponding boundary maps,

$$\cdots \xrightarrow{\partial_{p+2}} C_{p+1} \xrightarrow{\partial_{p+1}} C_p \xrightarrow{\partial_p} C_{p-1} \xrightarrow{\partial_{p-1}} \cdots .$$

Within a chain group, we have two different kinds of p-chains, cycles and boundaries. A p-cycle is a p-chain, c, with $\partial_p(c) = 0$, meaning it has empty boundary. The set of p-cycles is the kernel of the boundary map, $\ker(\partial_p)$. A p-boundary is a p-chain that is the boundary of a $p+1$-chain, i.e. for $c_p \in C_p$, $c_p = \partial c_{p+1}$ for some $c_{p+1} \in C_{p+1}$. The set of p-boundaries is the image of the boundary map, $\mathrm{im}(\partial_p)$. Note that the $p+1$-boundaries are a subgroup of the p-cycles. Now, the p-th homology group is formally defined as

$$H_p(\mathcal{K}) = \ker(\partial_p)/\mathrm{im}(\partial_{p+1}).$$

Further, the p-th Betti number is defined to be the rank of the p-th homology group and is denoted β_p. Intuitively, β_p can be thought of as the number of p-dimensional features in $\mathcal{K}$.

2.2 Persistent Homology

Homology is a useful tool for studying a static topological space; persistent homology is a method of using homology to instead study a changing, parameterized space called a filtration. A filtration

[1] We assume finiteness throughout this entire dissertation.

is a nested set of simplicial complexes,

$$K_0 \subseteq K_1 \subseteq K_2 \subseteq \cdots \subseteq K_n. \tag{2.1}$$

Computing k-dimensional homology of each space in the filtration, the inclusions in (2.1) induce linear maps between the homology groups,

$$H_k(K_0) \to H_k(K_1) \to \cdots \to H_k(K_n). \tag{2.2}$$

By studying these maps, we study how the homology of the space changes through the filtration. In particular, we care about when features appear and disappear in this sequence. We say a k-dimensional feature γ is "born" at the i-th step of the filtration if $\gamma \in H_k(K_i)$ but $\gamma \notin H_k(K_{i-1})$. In other words, γ is born at i if $\gamma \notin \text{Im}\,(H_k(K_{i-1}) \to H_k(K_i))$. A feature "dies" at the j-th step of the filtration if it merges with an older feature going from K_{j-1} to K_j.

A persistence diagram is a way of representing the births and deaths of homology classes. Formally, a persistence diagram, D, is defined as a collection of points, D, given by

$$D = \{(x, y) \in \mathbb{R}^2 \mid 0 < x < y\}.$$

A point $(b, d) \in D$ represents a class in the persistence module that was born at b and died at d. Persistence diagrams are typically visualized as scatter plots of the points in D. This is called the birth-death plane, or birth-death coordinates. In a filtration, a feature is born before it dies, so all persistence points will be above the diagonal through the birth-death plane. Another popular modification of a persistence diagram is to plot a class that is born at b and dies at d as the point $(b, d - b)$ where the quantity $d - b$ represents how long a feature lived, referred to as its "lifetime." This is called the birth-lifetime plane, or birth-lifetime coordinates.

2.2.1 Persistent Homology of Point Cloud Data

There are many ways of defining a filtration on different types of data such as point clouds, images, and graphs. Here we will introduce the one method, however additional methods will be introduced

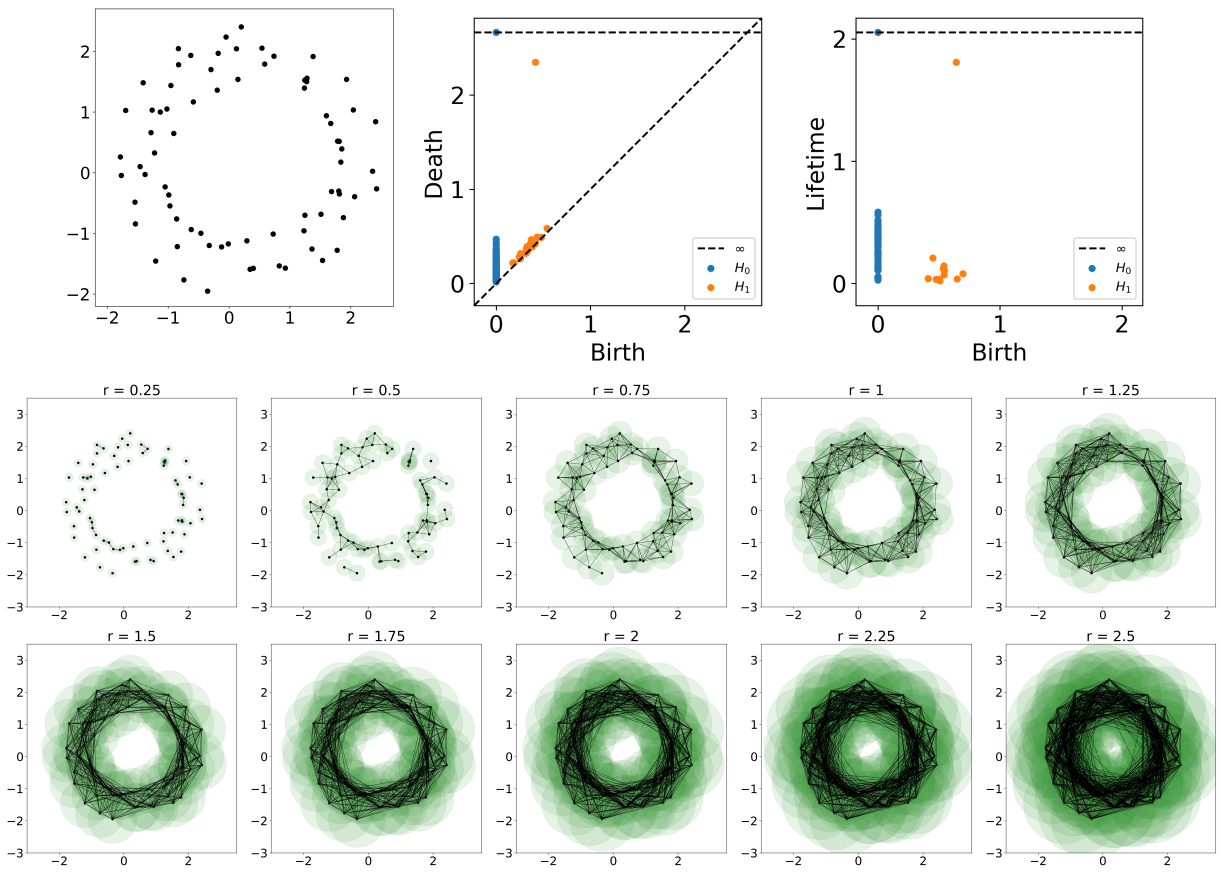


Figure 2.1: Examples of a point cloud, the Vietoris-Rips complex on several scales and the resulting persistence diagram in birth-death coordinates and birth-lifetime coordinates.

in the relevant chapters where they are needed. The Vietoris-Rips complex, or Rips complex for short, is a method of creating a simplicial complex from a point cloud. Here, we need only assume that a point cloud is a collection of points with a notion of distance; however, in practice, this distance often arises from a point cloud in Euclidean space inheriting the ambient metric. Given a point cloud X and a distance r the Vietoris-Rips complex $R(X, r)$ is a simplicial complex where for every finite set of n points with maximum pairwise distance at most r, the $n-1$ simplex formed by those points is included in $R(X, r)$. This can be visualized as centering a ball of radius $\frac{r}{2}$ on each point in X. When two of balls intersect, an edge is added between those points, and when there are three sets of pairwise intersections, a triangle is added between the three vertices. These complexes have the property that if $r_i < r_j$ then $R(X, r_i) \subseteq R(X, r_j)$. Thus, for any increasing set

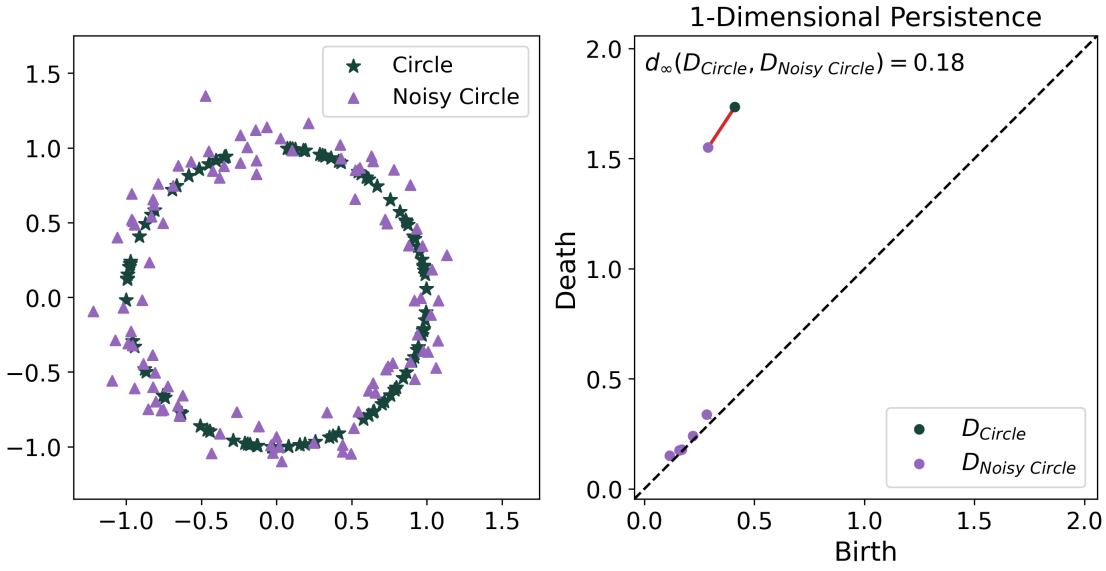


Figure 2.2: Two example point clouds and their corresponding 1-dimensional persistence diagrams with matching shown as dashed lines. Figures generated using scikit-tda [3].

of distance values, $0 \leq r_0 \leq r_1 \leq r_2 \leq \cdots \leq r_n$, we get a filtration,

$$R(X, r_0) \subseteq R(X, r_1) \subseteq R(X, r_2) \subseteq \cdots \subseteq R(X, r_n). \tag{2.3}$$

An example of a point cloud, several steps in the Vietoris-Rips filtration, and the resulting persistence diagram are shown in Fig. 2.1. In general, we do not even need the coordinates of the points in the point cloud; all that is needed to compute the Vietoris-Rips filtration is a matrix of the pairwise distances between all points.

2.2.2 Metrics on Persistence Diagrams

The most common metrics used to compare persistence diagrams are the Wasserstein distance and bottleneck distance, which are closely related. First, let $\Delta = \{(c, c) \in \mathbb{R}^2\}$ be the set of points along the diagonal in persistence diagrams, representing features where the birth time equals the death time. Note that Δ can have infinitely many copies of each point along the diagonal. The p-Wasserstein distance ($p \geq 1$) between two persistence diagrams D_1, D_2 can be defined as

$$d_{W_p}(D_1, D_2) = \inf_{\varphi} \left(\sum_{x \in D_1} \|x - \varphi(x)\|_q^p \right)^{1/p} \tag{2.4}$$

where $\varphi : D_1 \cup \Delta \to D_2 \cup \Delta$ is a bijection between the persistence diagrams that allows points in the diagrams can be matched to the diagonal. An example of this bijection, frequently called a matching, can be seen in Fig. 2.2. In practice, typically $q = \infty$ however there is evidence that choosing $q = p$ is a better choice [30]. Similarly, the bottleneck distance is defined as

$$d_\infty(D_1, D_2) = \inf_{\varphi} \sup_{x \in D_1} \|x - \varphi(x)\|_\infty \tag{2.5}$$

where $\varphi : D_1 \cup \Delta \to D_2 \cup \Delta$ is again a bijection that allows matching to the diagonal. However, in practice, these metrics are very computationally expensive and often cannot be used in application to large datasets.

A more simple approach is to develop statistics that can be calculated on a single persistence diagram. The most common is maximum persistence, which is also sometimes called maximum lifetime. Given a persistence diagram, D, maximum persistence is defined as

$$\text{MaxPers}(D) = \max_{(b,d) \in D} d - b. \tag{2.6}$$

This value represents the maximum lifetime of all homological features in the data. Using maximum persistence as a statistic on persistence diagram has shown success in numerous applications [11, 12, 27, 31]. However it may not always be sufficient. Typically, points with longer lifetime are interpreted as the most important features in the data since they are the ones that persist through the filtration the longest, while points with smaller lifetimes are considered noise. However this is not always the right interpretation; it has been shown in that shorter bars are important in many applications [32, 33].

One important property of persistence diagrams is that they are stable, meaning that noise and small changes in the input data results in a small change in the persistence diagram [34, 35]. An example can be seen in Fig. 2.2. The persistence diagrams for the circular point cloud and for the noisy version of the point cloud are very similar and thus have a small bottleneck distance. However there are certain types of noise that are more problematic. While the purple triangles are slightly perturbed from the circle in Fig. 2.2, a point in the middle of the circle would more drastically change the persistence diagram.

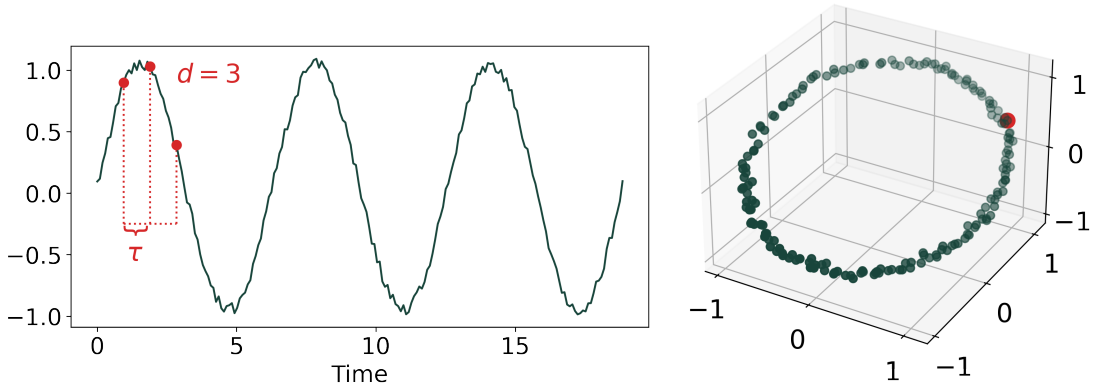


Figure 2.3: Example time series and corresponding time delay embedding using parameters $\tau = 10$ and $d = 3$.

2.3 Time Series Analysis

One common tool to analyze time series data is the time delay embedding. It is built off of underlying work in differential geometry and dynamical systems by Whitney [36] and Takens [4], respectively. The underlying assumption of this tool is that the time series is an observation of an underlying dynamical system. More formally, we assume the state space is a manifold, M, and the system is given by a smooth map, $f : M \to M$. Further, assuming the dynamics live on an attractor, $A \subset M$, then there exists a diffeomorphism $\phi : A \to \mathbb{R}^d$ for a well-chosen dimension, d. Whitney's embedding theorem says an m-dimensional manifold can be smoothly embedded in $\mathbb{R}^{2m}$. All together, this means that given an observation function of the underlying system $\alpha : M \to \mathbb{R}$ one can embed α into $\mathbb{R}^d$ in a way that the embedding is diffeomorphic (and thus topologically equivalent) to the original underlying attractor. This embedding is called the time delay embedding (also sometimes referred to as Takens' embedding or the sliding window embedding) and requires three inputs: the observation function $[x_1, \ldots, x_n]$, an embedding dimension d, and a delay (sometimes called a lag) τ. Sometimes these parameters are expressed as the window size, $d\tau$. The embedding is then the point cloud $\mathbf{X} = \{\mathbf{x}_i := (x_i, x_{i+\tau}, \ldots, x_{i+(d-1)\tau})\} \subset \mathbb{R}^d$. Figure 2.3 shows an example time series and depicts how the two parameters, d and τ, are used in the creation of the time delay embedding. The time series value of three red points along the time series correspond

to the three coordinates of the red point on the right side. In practice, given a time series, one does not know the underlying system, the manifold M, or the dimensional of the manifold. Thus, there is no way theoretically to determine the correct embedding dimension. However, there are several heuristics that work well in practice for selecting the delay and dimension parameters [37–43].

Most of these heuristics are iterative, testing various values of τ or d and determining which is best. For example one of the most common methods of selecting the dimension d is using false nearest neighbors [38]. Intuitively, this method works by assuming that if two points are near each other when embedded in dimension d, but then are far apart in dimension $d + 1$, then they were false nearest neighbors in dimension d, and thus dimension d is insufficient to embed into. This process would then compare embedding into dimension $d + 1$ and $d + 2$, and continue until the proportion of neighbors that are false is 0, or sufficiently small. In Chapter 5 we will introduce a heuristic for selecting the delay which we will use in experiments in that chapter.

For another example, consider the Lorenz system defined as,

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - z), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= xy - \beta z. \end{aligned} \tag{2.7}$$

We can see one example solution using the parameters, $\sigma = 10, \rho = 28$ and $\beta = 8/3$, using initial conditions $[1.0, 1.0, 1.0]$ in Fig. 2.4. This is generated using full knowledge of the Lorenz system, however if we only retain the x-coordinates over time, we get the time series shown in the bottom left plot of Fig. 2.4. Using the time delay embedding, we can reconstruct the topological structure setting $d = 3$ and $\tau = 8$, resulting in the bottom right plot in Fig. 2.4. While the time delay embedding is not identical to the original system, it still has two loop-like structures and is topologically equivalent.

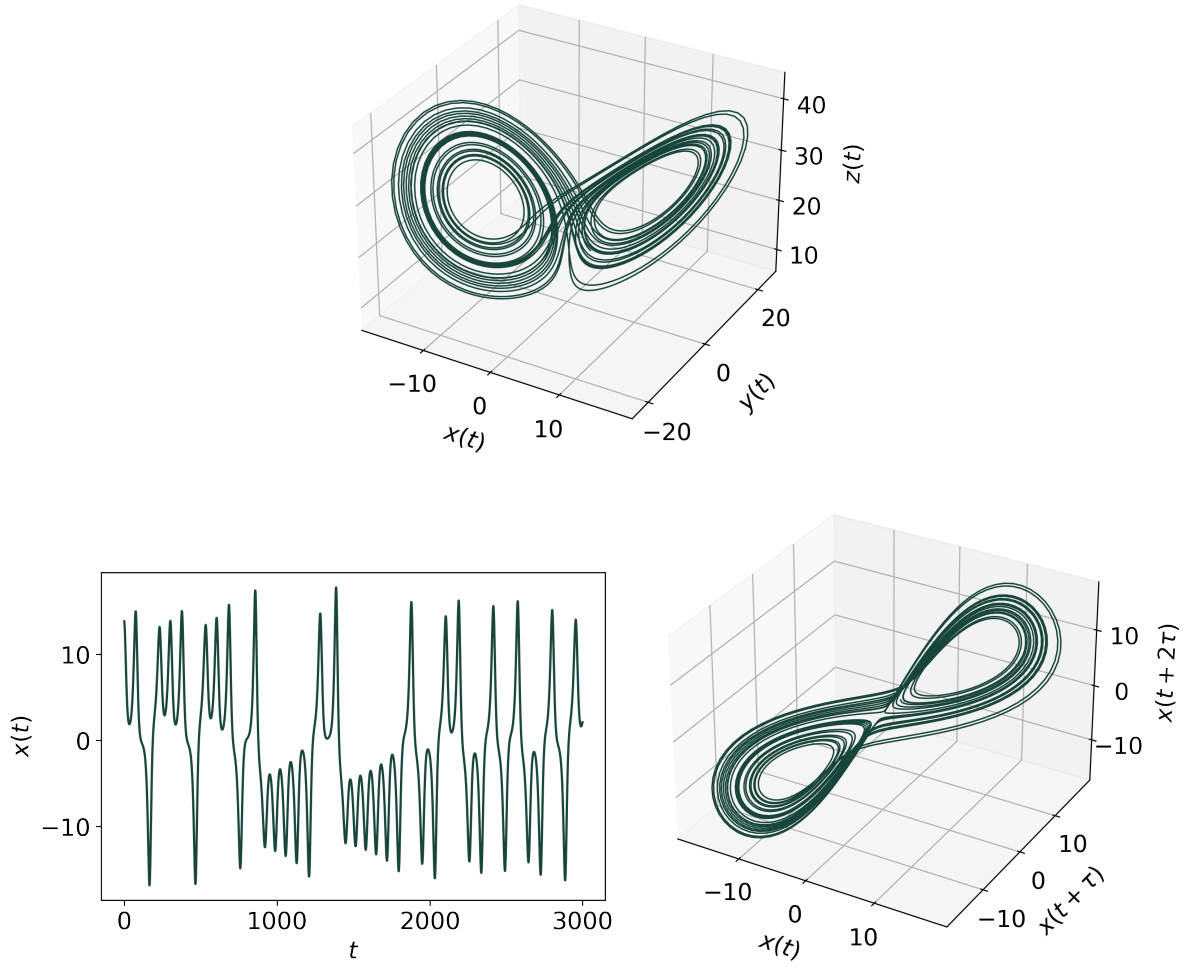


Figure 2.4: Top left: Example solution of the Lorenz system (as defined in Eqn. 2.7) using parameters, $\sigma = 10, \rho = 28$ and $\beta = 8/3$, using initial conditions $[1.0, 1.0, 1.0]$. Bottom left: Time series of x-coordinates from the example solution. Bottom right: Time delay embedding of the x-coordinate time series using $d = 3$ and $\tau = 8$.

2.4 Related Works

Significant work has been done using persistent homology on time delay embeddings. Specifically, in [5], the authors provide a full theoretical analysis of persistent homology for time series analysis. The authors show that maximum persistence of the persistence diagram computed from the time delay embedding can be used to quantify periodicity. Additionally, they provide a description of how persistence diagrams change depending on the embedding dimension and the delay parameter. This work provides sound theoretical backing for the use of persistent homology on time delay

embeddings.

This combination of tools has shown success in numerous application areas as well. In the literature so far, persistent homology has been shown to quantify features of a time series such as periodic and quasi-periodic behavior [5–10]. Existing applications in time series analysis include studying machining dynamics [11–14], gene expression [15], financial data [16], video data [17, 18], sleep-wake states [19, 20], motion tracks [21, 22], and the movement of C. elegans [23]. This list is far from exhaustive, but shows the wide range of fields that these tools have been successfully applied to.

In this dissertation, we will present applications of using topological tools to analyze time series data, as well as introduce new methods.

CHAPTER 3

QUANTIFYING A DIURNAL CYCLE IN A TIME SERIES OF HURRICANE IMAGERY

A time series can consist of different types of data. While we typically think of a time series as a single variable recorded over some time period, we can also think of a video or a movie as a time series of images. Satellite imagery is one example of a source of such data, where, for example, the satellite records temperatures of the Earth in a certain region. This type of data is especially useful in the field of atmospheric science to study phenomena such as hurricanes.

A recently discovered pattern exhibited in many major hurricanes is a diurnal cycle that is apparent in changing cloud temperatures within a hurricane [44]. This cycle, formally called the tropical cyclone (TC) diurnal cycle, can be seen as a cyclical pulse in the cloud field that propagates radially outward from the storm's center. The pulses begin forming in the inner core of the storm, appearing as a region of cooling cloud-top temperatures in the satellite imagery. The area of cooling takes on a ring-like structure as cloud top warming occurs on the inside edge as the cooling moves away from the storm center. Figure 3.1 shows two examples of the ring-like structure in Hurricane Felix. This cycle is of interest to atmospheric scientists as it has implications for the storms structure and intensity. However, as methods to detect this cycle have thus far been mostly qualitative in nature, we seek a method of automatic and quantifiable detection of this cycle.

Because this cycle can be seen as an expanding circular structure, 1-dimensional persistent

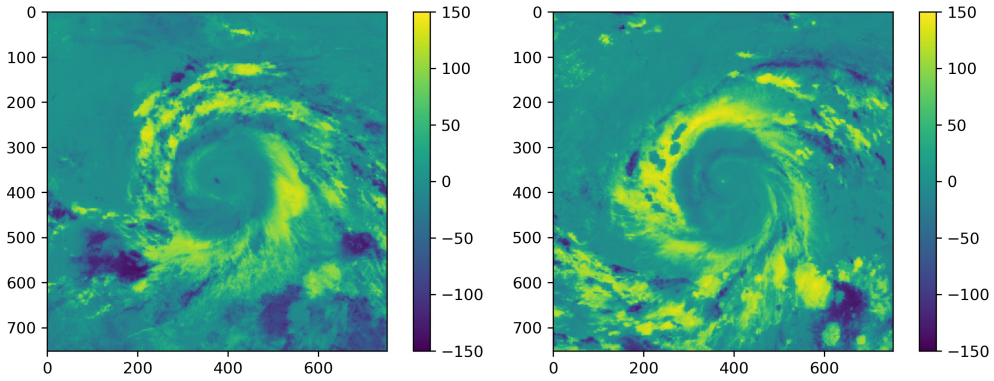


Figure 3.1: Two examples of the ring-like structure of the tropical cyclone diurnal cycle.

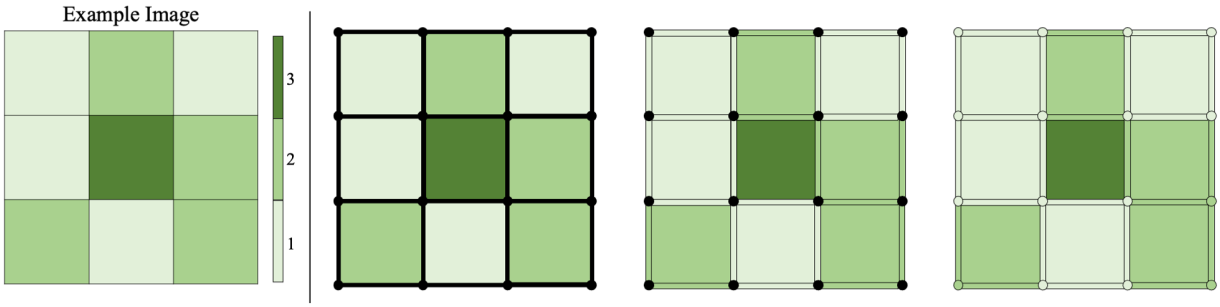


Figure 3.2: Example image and visual on propagating greyscale function values to lower dimensional cubes.

homology seems like an obvious choice of tool for detecting the diurnal cycle. We developed a method of utilizing persistent homology to detect and quantify this cycle using a time series of infrared (IR) satellite imagery data. The work in this chapter was published in [27].

3.1 Persistent Homology on Images

Persistent homology is a versatile tool in that it can be applied to a filtration resulting from many different types of data. We saw the case of simplicial data in Sec. 2.2, however, in the case of images, cubical complexes are a more natural choice. Here we introduce cubical complexes following the presentation in [45].

An elementary interval in $\mathbb{R}$ is a closed interval of the form $[l, l+1]$ or $[l]$, $l \in \mathbb{Z}$ and is called degenerate or nondegenerate respectively. An elementary cube is the product of elementary intervals, $Q = I_1 \times I_2 \times \cdots \times I_d \subset \mathbb{R}^d$. The dimension of an elementary cube is defined as the number of nondegenerate intervals in Q. A complex, $\mathcal{K}$, is cubical if it can be written as a finite union of elementary cubes. As with simplicial complexes, we have a notion of faces: if P and Q are elementary cubes and $P \subseteq Q$, then P is a face of Q (denoted $P \leq Q$). Intuitively, this construction is similar to a simplicial complex, however triangles and higher dimensional analogues, are replaced by elementary cubes. To avoid confusion, we will call the building blocks of simplicial complexes p-simplices, and the building blocks of cubical complexes p-cubes.

Given an greyscale image, we can construct a filtered cubical complex where each pixel represents a 2-cube with function value equal to the greyscale value of that pixel. Then function values

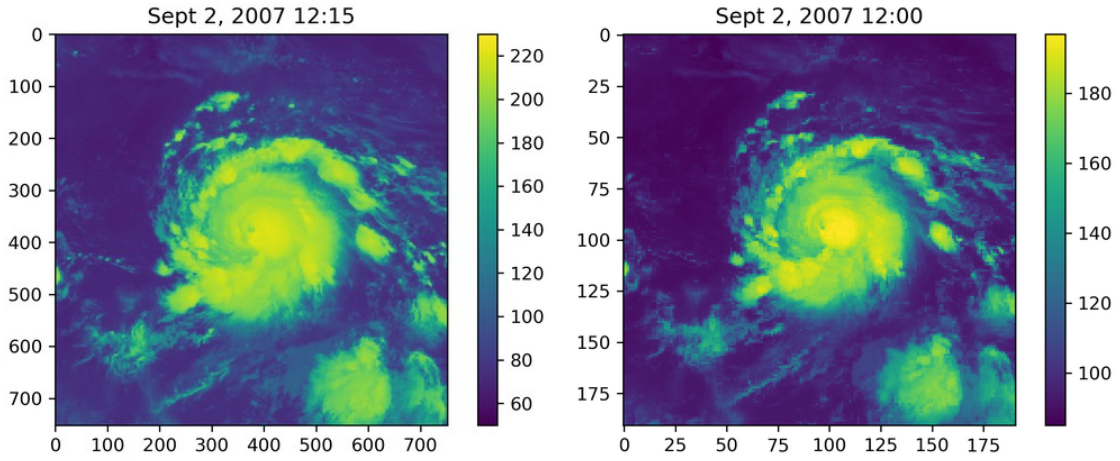


Figure 3.3: Original satellite imagery from Hurricane Felix GOES-12 data set (left) and Hurricane Felix GridSat-GOES data set (right) at approximately the same time.

can be propagated down to lower dimensional faces. More formally,

$$f(Q) = \min_{P \in \mathcal{K} \text{ such that } Q \leq P} f(P)$$

For example, the function value on an edge is equal to the minimum of the pixel values adjacent to it, and similarly for vertices. An example of this can be seen in Fig. 3.2.

Given the cubical complex with a function $f : \mathcal{K} \to \mathbb{R}$, one can define a filtration formed from nested sublevel sets,

$$f^{-1}(-\infty, \alpha_0] \subseteq f^{-1}(-\infty, \alpha_1] \subseteq \cdots \subseteq f^{-1}(-\infty, \alpha_n]$$

for increasing parameter value $\alpha_0 \leq \alpha_1 \leq \cdots \leq \alpha_n$ with all $\alpha_i \in \mathbb{R}$. This forms a filtration as defined in Eqn. 2.1 and thus, persistent homology can be computed as described in Sec. 2.2. The sublevel set filtration[1] is very commonly used on images in this way, and is what we will use for the hurricane images.

3.2 Imagery Data

For this study, we will focus on two hurricanes that exhibit the diurnal cycle behavior, Hurricane Felix from 2007 and Hurricane Ivan from 2004. We worked with two types of geostationary

[1]Technically, most filtrations can be viewed as a sublevel set filtration that only differ in the definition of the $\mathbb{R}$-valued function and choice of domain. The Vietoris-Rips filtration is a sublevel set filtration defined on the complete simplicial complex, where the real valued function is induced by the pairwise distances.

operational environmental satellite (GOES) data, each with varying spatial and temporal resolution. The first type (hereafter referred to as the GOES-12 data sets) consists of data in hourly increments, with the exception of the 0400 and 0500 UTC each day (due to the GOES-12 satellite eclipse period). This imagery has a spatial resolution of 2 km^2 and each image covers a total area of approximately 1500 km $\times$ 1500 km, represented as a 752 $\times$ 752 matrix. The second type is the GridSat-GOES [46] data set and consists of data in 3-hour increments with the exception of 0600 UTC each day. Each pixel has a resolution of 8 km^2 and each image is cropped to a 191 $\times$ 191 matrix to approximately match the area covered by the first set of data. The cropped version covers a total area of approximately 1530 km $\times$ 1530 km. For both data sets, the images are storm-centered, meaning the center of the hurricane is in the middle of the image. This is done to ensure the hurricane is aligned at each point in time.

For Hurricane Felix, we studied both types of data sets to test the flexibility of the method across spatial and temporal resolution. The Felix GOES-12 data set spans 2 to 4 September 2007, while the Felix GridSat-GOES data set spans spanning 31 August to 6 September 2007. For Hurricane Ivan, we only used the GOES-12 data set, which spans 30 August to 1 September 2004. Figure 3.3 shows example satellite imagery for both Hurricane Felix data sets.

3.3 Method of Detection and Quantification

Our method of detecting and quantifying periodic circular structure in IR satellite imagery combines existing methods from the fields of image processing, topological data analysis, and signal processing. In this application, it is not only the structure at a given time that matters, but also how the structure changes over time.

Initially, we have a time series of IR satellite images, represented as a matrix of pixel values $S(t)$ for time t. The method works by applying four steps to these matrices, examples of which can be seen in Fig. 3.4.

1. For all times t, given the original brightness temperature image $S(t)$, we compute the six-hour differences, $M(t) = S(t + 6) - S(t)$.

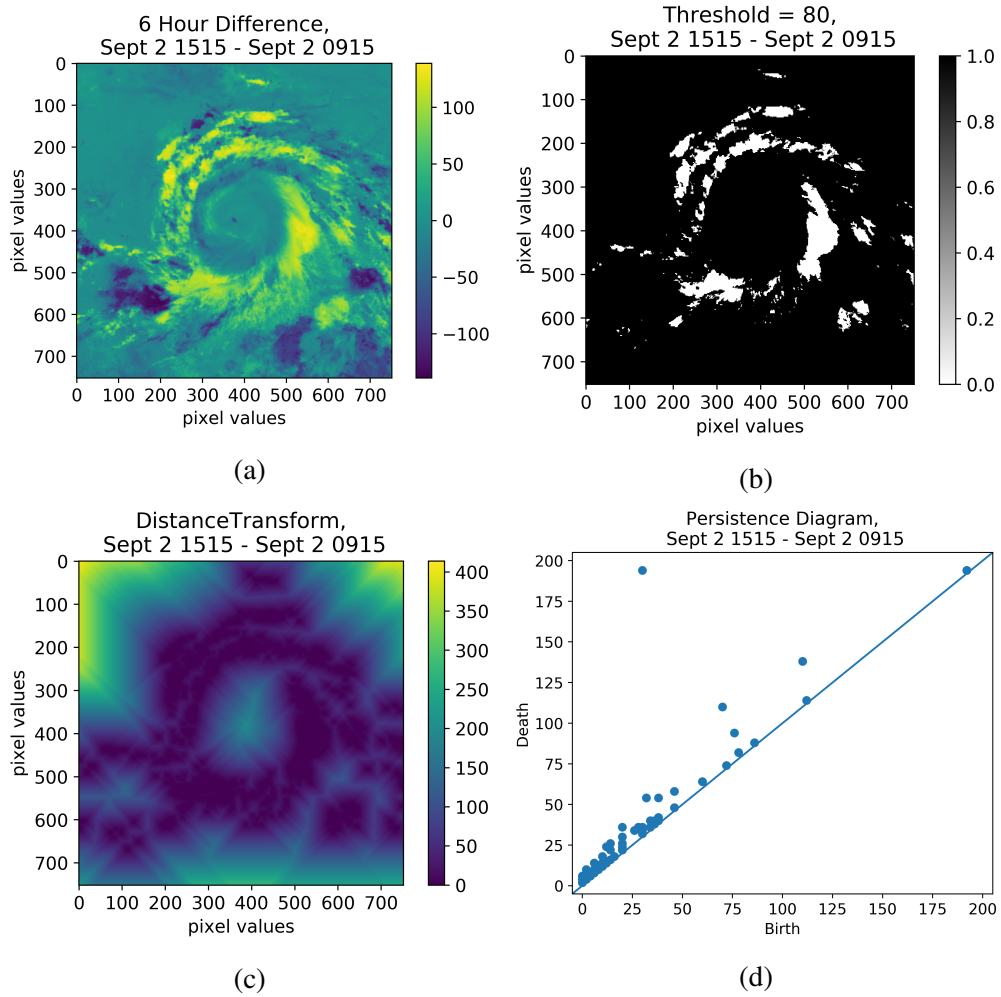


Figure 3.4: (a) Example of 6 hour difference, $M(t)$, from the Felix GOES-12 data set; (b) Thresholded subset, $M(t)_\mu$ where $\mu = 80$; (c) Distance transform function applied to $M(t)_\mu$; (d) Corresponding sublevel set persistence diagram. These figures correspond to steps 1–4 in 3.3.

2. Fix a threshold μ and let $M(t)_\mu$ be the subset of $M(t)$ which has function value less than μ:

$$M(t)_\mu[i,j] = \begin{cases} 1 & \text{if } M(t)[i,j] < \mu \\ 0 & \text{otherwise.} \end{cases}$$

3. To each matrix $M(t)_\mu$, apply the distance transform [47, 48], which gives a new matrix $D(t)$, defined as

$$D(t)[i,j] = \min d(m_{i,j}, x)$$

where $m_{ij} = M(t)_\mu[i,j]$, x is a 0-valued pixel and d is the L_∞-distance. This gives a new greyscale image from which a cubical complex can be constructed.

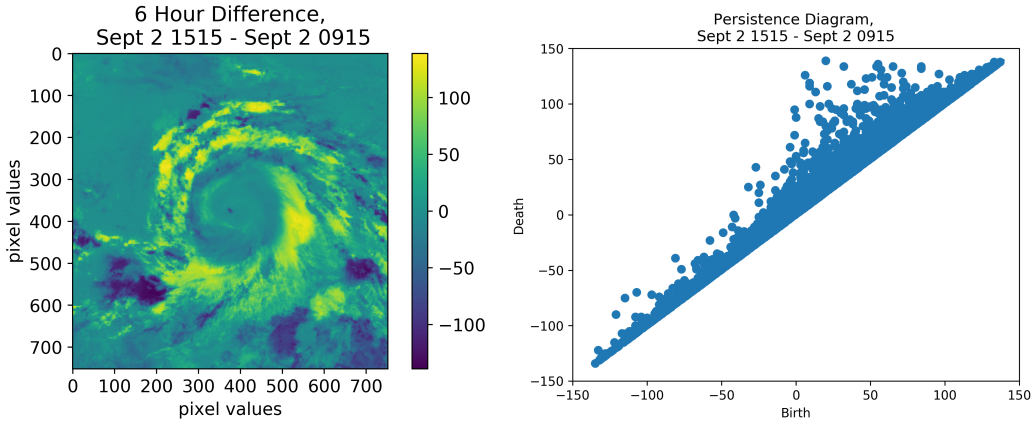


Figure 3.5: Example of 6-hour difference image and the corresponding persistence diagram.

4. Compute sublevel set persistence on the function $D(t)$ using the `cubtop` method in Perseus [49, 50], which calculates persistent homology for cubical complexes using concepts from discrete Morse theory.

The first three steps are modifications to the binary image, while the last step constructs a cubical complex as described in Sec. 3.1. Since the TC diurnal pulse is a cooling ring propagating outward through the day, step 1 is necessary in order to see the changes in the GOES satellite brightness temperature. In previous work by atmospheric scientists, the 6 hour difference was found to be most effective difference to detect the cycle [44, 51]. While one can compute persistence directly from the 6-hour differences where the circular features are visually prominent in the data, the results did not show any relevant features. This discrepancy is due to the extreme differences in the function values between the circular sections which prevents the sublevel sets from containing the full circular structure until very late in the filtration. As seen in Fig 3.5, the persistence diagram computed directly on a 6-hour difference image has no significant off diagonal point, thus it is not detecting a significant circular feature despite the visible feature in the image. Thus, the addition of steps 2 and 3 were necessary to detect the circular structure that we see visually. For step 2, all the examples shown use a threshold of $\mu = 80$, however we will address the choice of threshold in Sec. 3.4.1.

This thresholding gives a binary image, however the persistent homology of binary images

is uninteresting as there would only be two steps in the filtration and likely would not detect the circular structure. The distance transform creates a new greyscale image where pixel values are based on distance to the thresholded region. This creating a "blurring" effect where all pixels that are close to a white pixel have a smaller greyscale value, and the further away the pixel it is the higher the value. This sets up the image perfectly for sublevel set persistence. The distance transform has been used in conjunction with persistent homology in applications including porous materials [52–57]. However, this is the first time to our knowledge that this combination of tools has been applied in atmospheric science.

After these four steps are applied to each image, we calculate maximum persistence of each persistence diagram as defined in Eqn. 2.6. By plotting maximum persistence over time, we can see how the most prominent circular feature changes through the progression of the day and life of the TC. This plot should show an oscillatory pattern, detecting the change in the diurnal cycle throughout the day. In order to quantify this oscillatory pattern, we use the discrete Fourier transform [58]. Using the power spectrum, we pick the frequency corresponding to the highest peak for each data set, which gives the frequency of the most prominent periodic signal in the data. Additionally, we use the inverse Fourier transform to see how closely this signal matches with the maximum persistence time series.

3.4 Results

After the data is prepared, we apply the steps described in Sec. 3.3 to each data set. For the two Hurricane Felix data sets, as they are from the same hurricane, we would expect the results to be similar despite the temporal and spatial resolution differences. Plotting the calculated maximum persistence over time, we get the time series plotted as solid lines in Fig. 3.6. The plots show an oscillatory pattern for all three data sets which appears to repeat approximately daily.

To verify the periodicity of the oscillatory pattern, we apply the discrete Fourier transform and calculate the power spectrum for each data set. Each power spectrum is shown in Fig. 3.7. The maximum peaks in the power spectra give a frequency of $f = 0.976$ cycles per day for the Felix

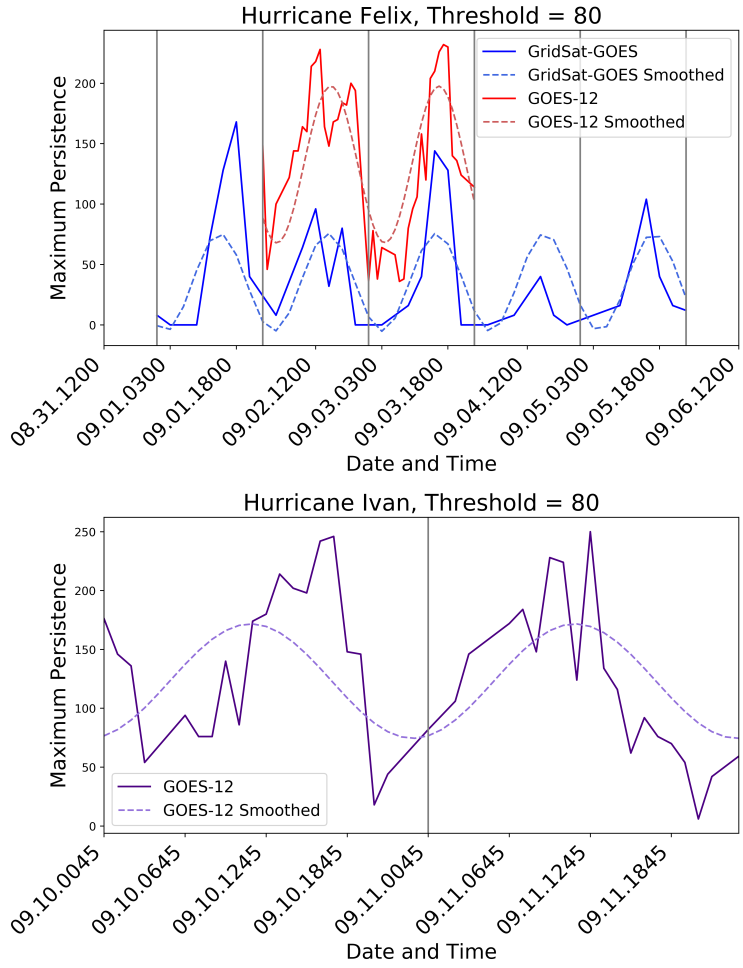


Figure 3.6: Maximum persistence plotted over time for both hurricane Felix data sets (left) and the hurricane Ivan data set (right) using threshold $\mu = 80$ in addition to the reconstructed versions, created using inverse Fourier transform. Gray vertical lines separate days according to UTC. The labels on the x-axis are formatted as MM.DD.time.

GridSat-GOES data set, $f = 0.979$ cycles per day for the Felix GOES-12 data set, and $f = 1.0$ cycles per day for the Ivan GOES-12 data set. We use this frequency, f, to calculate the period, T, of the cycle by calculating

$$T = 24/f,$$

giving the period of the sinusoid in hours per cycle. Doing so gives the result that the cycle is repeating every 24.6 hours for the Felix GridSat-GOES data set, every 24.5 hours for the Felix GOES-12 data set, and 24.0 hours for the Ivan GOES-12 data set.

Using the most prominent frequency for each data set, we calculate the inverse Fourier transform,

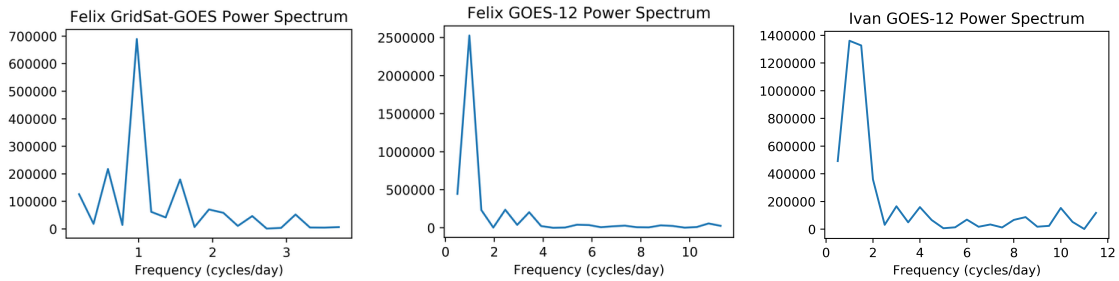


Figure 3.7: Power spectrum for the Felix GridSat-GOES data set (left), the Felix GOES-12 data set (middle), and the Ivan GOES-12 data set (right).

and plot these reconstructed sinusoids over the original data. These sinusoids, plotted as the lighter dashed lines in Fig. 3.6, closely resemble the patterns exhibited by the original maximum persistence versus time plots; therefore, these approximately 24 hour patterns visible in the plots are also detected mathematically, which verifies the claim that our method is detecting a daily cycle in each data set. Additionally, since for both Hurricane Felix data sets, the plots of maximum persistence against time seem to match and both have similar detected periodicity from the discrete Fourier transform, our method is robust to the temporal and spatial resolution differences in these two data sets.

3.4.1 Choice of Threshold

The method described involves a choice of threshold, so we used a variety of thresholds, $\mu \in \{25, 30, \ldots, 100\}$, to test the sensitivity of our method to the parameter choice. For both data sets from Hurricane Felix, our method is very robust to the choice of threshold. In Fig. 3.8, the top row are plots that represent maximum persistence versus time for the Hurricane Felix data sets using a variety of thresholds. There is a clear periodic pattern for both data sets across most of the thresholds shown. In fact, for all thresholds tested the Fourier transform detects a period of 24.6 hours and 24.5 hours for the Felix GridSat-GOES data set and GOES-12 data set respectively. For $\mu < 35$ and $\mu > 90$ the Fourier transform is unable to pick up the daily pattern in the Felix GridSat-GOES data set.

For Hurricane Ivan, the plot is shown on the bottom row of Fig. 3.8 for thresholds $\mu \in$

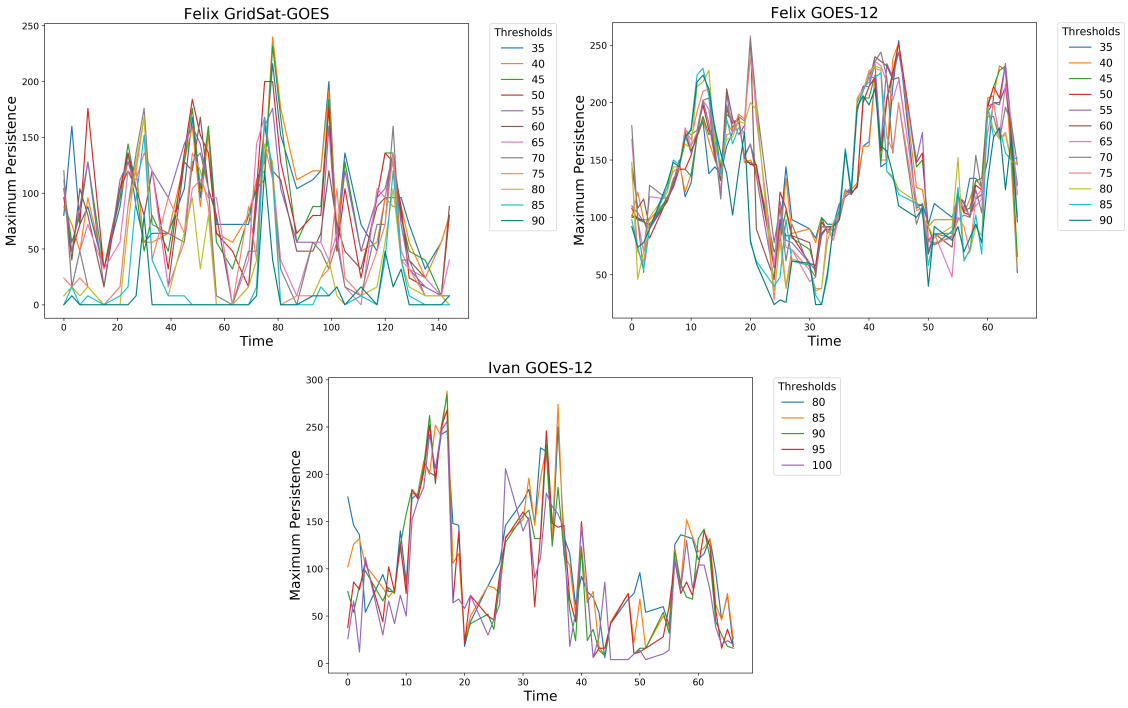


Figure 3.8: Maximum persistence vs time plot for Hurricane Felix (top row) and Hurricane Ivan (bottom row). Hurricane Felix results are shown for all thresholds $\mu \in \{35, 40, \ldots, 90\}$ while Hurricane Ivan results are shown for $\mu \in \{80, 85, 90, 95, 100\}$.

$\{80, 85, 90, 95, 100\}$. For all of the threshold values shown, the Fourier transform consistently detects a 24.0 hour period in the maximum persistence values. This is a smaller range of threshold values than those that detect a daily cycle in Hurricane Felix, but for thresholds $\mu \in \{80, 85, 90\}$, our method detects a daily cycle in all three data sets. Thus, the method may require some parameter tuning, but our analysis of these three data sets gives a range of values to start with when testing new data sets.

3.4.2 The Influence of Noise

While the above method detects a daily cycle, there are some instances where the six-hour difference introduces noise because of varying behavior in the center of the hurricane. Figure 3.9 shows an example of how this noise can appear in the thresholded image and the distance transform for the Felix GOES-12 data set. A small area of pixels above the threshold cause the distance transform to fill in the center of the circular region, thus potentially changing the value of maximum persistence.

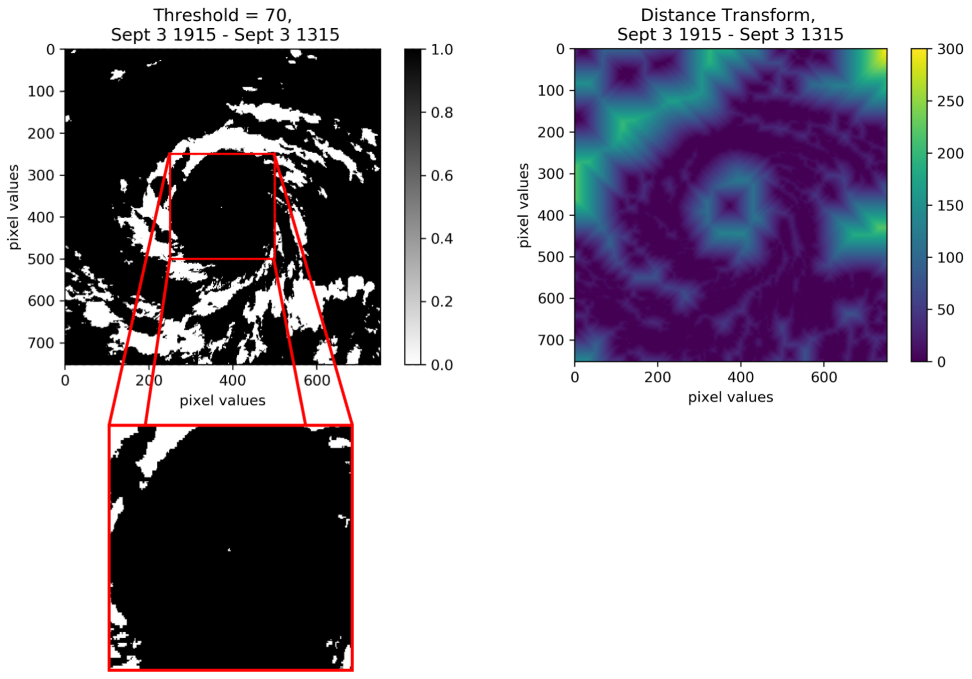


Figure 3.9: Example of a thresholded image with noise and the resulting distance transform image.

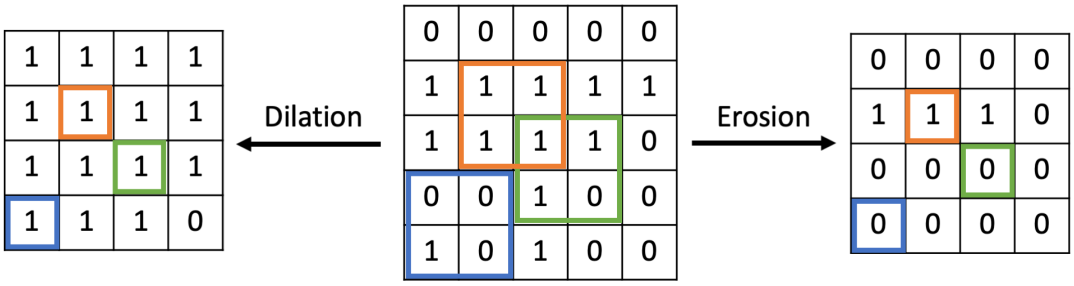


Figure 3.10: Example binary image and the result after erosion and dilation with a 2×2 square kernel.

Note that while persistence diagrams are stable, with this type of noise, stability does not guarantee small changes in the persistence diagrams. To combat this, before applying the distance transform, we use a method from mathematical morphology [59] called opening to de-noise the image and see how this impacts the detected periodicity.

Opening is the combination of two tools from mathematical morphology: erosion and dilation [59]. Both involve moving a structural element through a binary image and adding or removing to the foreground of the image (where the foreground is the portion of the binary image with value 1). In general, any structural element can be used however in this case we will use an $n \times n$ square.

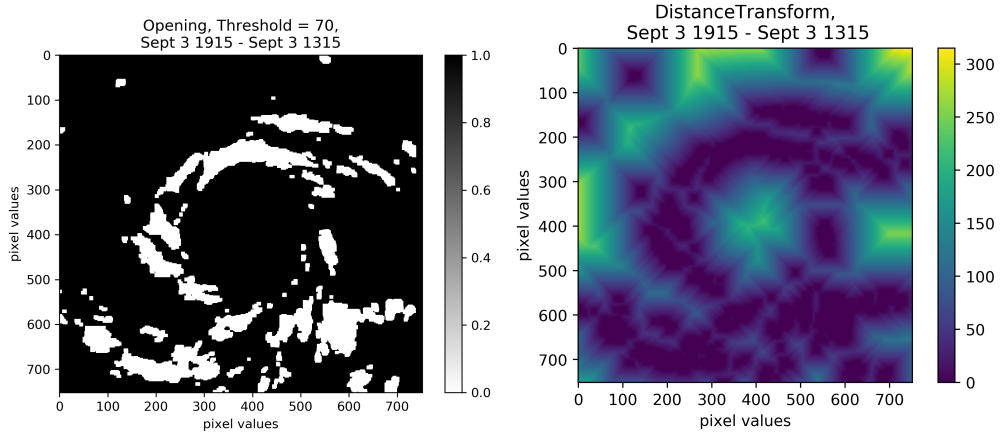


Figure 3.11: Example of a thresholded image after opening is applied and the corresponding distance transform.

As in Sec. 3.3, we can treat a binary image as a matrix, B. Erosion of B, which we will denote as $E(B)$ can be defined as

$$E(B)[i,j] = \begin{cases} 1 & \text{if } \forall\ k \in \{i,\ldots,i+(n-1)\}, \ell \in \{j,\ldots,j+(n-1)\}, B[k,\ell] = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, dilation of B, denoted $D(B)$, can be defined as

$$D(B)[i,j] = \begin{cases} 1 & \text{if } \exists\ k \in \{i,\ldots,i+(n-1)\}, \ell \in \{j,\ldots,j+(n-1)\} \text{ such that } B[k,\ell] = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Note that both of these operations result in a smaller binary image as $n-1$ rows and columns are removed. Figure 3.10 shows an example of dilation and erosion using a 2×2 square structural element. Opening of a binary image, B, is erosion followed by dilation, $D(E(B))$, which will remove noise and rebuild the area around the boundary.

We apply opening to the binary thresholded image using a 8×8 pixel kernel for the GOES-12 data sets and a 2×2 pixel kernel for the GridSat-GOES data set to remove noise such as these center pixels. Note the difference in size of the kernel is due to the differences in spatial resolution between the two data sets. We use the python library `OpenCV` [60] for these computations. Opening is specifically implemented using the function `cv2.morphologyEx` using `cv2.MORPH_OPEN` as the second input. Figure 3.11 show the result when opening is used on the thresholded matrix and then

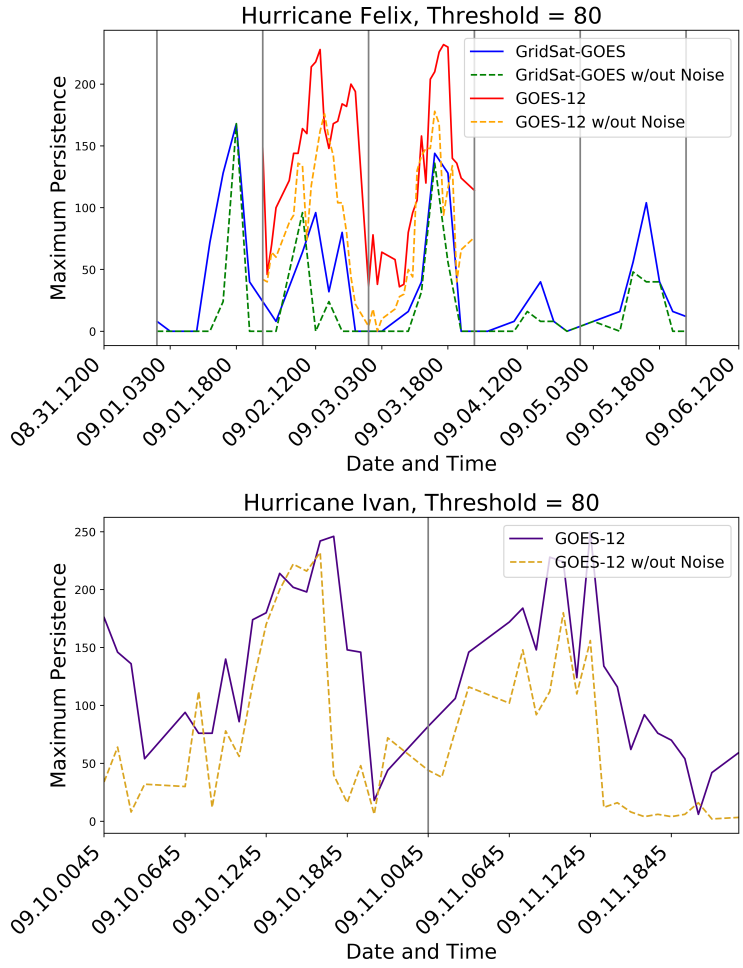


Figure 3.12: Maximum persistence plotted over time for all data sets using threshold $\mu = 80$ in addition to the versions using opening to remove noise. Gray vertical lines separate days according to UTC.

the distance transform is applied. Since the distance transform is no longer filled in, the opening process has removed the noisy pixels causing the issue.

Using this extra step in the method, we recalculate maximum persistence for all times and compute the estimated period of the new maximum persistence values using Fourier transforms. Figure 3.12 shows maximum persistence plotted versus time using our original method described in Sec. 3.4, and the method including the additional opening step. While the new maximum persistence values vary a little from the originals, the general oscillatory behavior seems similar. For both the Felix and Ivan GOES-12 data sets, the Fourier transform still detects a 24.5 and 24.0 hour cycle respectively. Thus the presence of noise in these data sets is not impacting the results.

However, for the Felix GridSat-GOES data set, the Fourier transform now detects a 15.375 hour cycle, likely due to the difference in spatial resolution. The GOES-12 data has higher spatial resolution, so applying opening to remove noise does not impact the global circular structure. The GridSat-GOES data has lower spatial resolution, and is therefore more sensitive to noise in the image. Thus, our method is more reliable when applied to higher spatial resolution data, and should be used with caution on lower quality data.

CHAPTER 4

TIME SERIES CLASSIFICATION USING ADAPTIVE TEMPLATE FUNCTION FEATURIZATION

In the previous section, we analyzed time series of images, resulting in a time series of maximum persistence values. However, in other cases you may have multiple time series that you wish to classify in some way. For example, if you have a time series of vibration data from a machining process, you may want to classify whether the machine is undergoing excessive vibrations causing chatter, a phenomena that is damaging to the machine pieces as well as to the material the machine is cutting [11, 61].

While combining persistent homology with machine learning sounds like a reasonable idea, the space of persistence diagrams is problematic. It is not a Banach space and does not have unique means or geodesics [30, 62, 63]. Thus, applying machine learning to persistence diagrams takes some additional mathematical creativity. One method of doing so is using a kernel function, where you compute a similarity matrix on the collection of persistence diagrams and use any kernel based machine learning method. Another collection of methods are featurizations, or methods of mapping from the space of persistence diagrams to Euclidean space in a way that maximizes the structure preserved. Then the resulting feature vectors can be used in any machine learning framework. Intuitively, we want persistence diagrams that are "close" in the space of persistence diagrams to have vectors that are "close" in the feature space. However, it has been shown that there is no isometric embedding from the space of persistence diagrams into Euclidean space [64, 65]. This means there is no feature map that will preserve the original metric, so while these methods are useful in practice, the theoretical guarantees are limited.

Numerous kernels [2, 66–72] and featurization methods [73–81] have been developed as the interest in using persistence diagrams for machine learning has grown in popularity. For brevity, here I will only focus on one featurization method, called the template function featurization [1], however surveys of additional methods can be found in [82, 83]. Further, I will present an adaptive

version of the template function featurization. The content of this chapter was published in [28].

4.1 Template Function Featurization

A template function is defined as any function on $\mathbb{R}^2$ that that is continuous, and has compact support contained within the upper half plane[1], $\mathbb{W} := \mathbb{R} \times \mathbb{R}_{>0}$. Let $\mathfrak{D}$ denote the space of all persistence diagrams. A template function $f : \mathbb{W} \to \mathbb{R}$ can be turned into a function on persistence diagrams as follows. Given a diagram in birth-lifetime coordinates, D, the function, $\nu_f : \mathfrak{D} \to \mathbb{R}$ is evaluated on each point in the diagram, and then summed, giving

$$\nu_f(D) = \sum_{\mathbf{x} \in D} f(\mathbf{x}).$$

A collection of template functions, $\mathcal{T}$, is called a template system if the resulting functions on persistence diagrams, $\mathcal{F}_\mathcal{T} = \{\nu_f : f \in \mathcal{T}\}$ separate points. That is, for every pair of diagrams, D and D', there exists a function $f \in \mathcal{T}$ such that $\nu_f(D) \neq \nu_f(D')$. As a true template system is infinite, vectorization is done by returning $\left(\nu_{f_1}(D), \cdots, \nu_{f_k}(D)\right)$ for functions in some subset of the template system. This is well justified since any function on persistence diagrams can be approximated by some reasonably chosen finite subset of a template system; see [1, Thm. 29]. In this paper, we will use two examples of template systems as given in [1]: tent functions and interpolating polynomials.

4.1.1 Tent Functions

Tent functions are an example of template functions that are meant to probe small regions of the persistence diagram. Again, recall everything is defined in the birth-lifetime plane. Given a point $\mathbf{a} = (a, b)$, and a radius $\delta \in \mathbb{R}_{>0}$ with $0 < \delta < b$, the tent function is defined to be

$$g_{\mathbf{a},\delta}(x, y) = \left| 1 - \frac{1}{\delta} \max\{|x - a|, |y - b|\} \right|_+,$$

where $|\cdot|_+$ denotes the positive value of the function, and 0 otherwise. This function is supported on the compact box $[a - \delta, a + \delta] \times [b - \delta, b + \delta]$, evaluates to 1 at $\mathbf{a}$, and decreases linearly to 0

[1]We will use birth-lifetime coordinates as described in Sec. 2.2 throughout this section, so all points in the persistence diagram lie in the upper half plane, rather than above the diagonal.

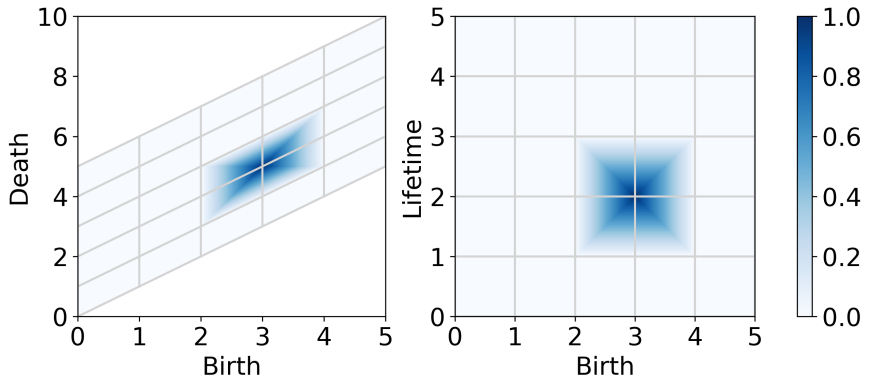


Figure 4.1: Example tent function, $g_{(3,2),1}$, drawn in the birth-death plane and birth-lifetime plane with $d = 5$, $\delta = 1$ and $\epsilon = 0$. Plot adapted from [1, Fig.4].

on the boundary of the box. An example of a tent function is shown in Fig. 4.1. Note that since the box must be compactly supported on persistence diagrams, the bottom edge of the box cannot lie on or below the x-axis. Given a persistence diagram $D = \{\mathbf{x} = (b_i, l_i)\}$, the tent function is the sum of the evaluation of this function on all points in the diagram,

$$G_{\mathbf{a},\delta}(D) = \sum_{\mathbf{x}\in D} g_{\mathbf{a},\delta}(\mathbf{x}).$$

The full template system consists of all tent functions $g_{\mathbf{a},\delta}$ which have compact support contained in $\mathbb{W}$. However, in practice, we work with the subset of these tent functions

$$\mathbf{G} = \{G_{(\delta i,\delta j+\epsilon),\delta} \mid 0 \leq i \leq d, 1 \leq j \leq d\} \tag{4.1}$$

by choosing the grid size, d, and a vertical shift, $\epsilon > 0$, to ensure G is compactly supported inside $\mathbb{W}$. This gives a $d \times (d+1)$ feature vector. In Fig. 4.1, the grid represents the mesh on which tent functions can be centered and a single tent function, centered at $(3, 2)$ with $\delta = 1$ and $\epsilon = 0$ is shown.

4.1.2 Interpolating Polynomial Functions

The second template system we work with are interpolating polynomials. Unlike the localized tent functions, interpolating polynomials have support that fills out the space, however to satisfy the properties of template functions, they will be transformed to have compact support. Given a mesh

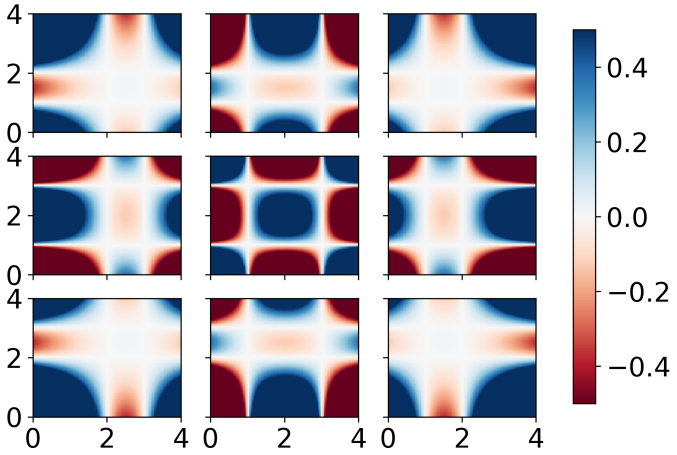


Figure 4.2: Examples of interpolating polynomials for the meshes $\mathcal{A} = \mathcal{B} = \{1, 2, 3\}$ where the plot drawn at (i, j) shows the polynomial, $p_{i,j}$, where $p_{i,j} = 1$ and 0 on all other mesh points. Plots adapted from [1, Fig. 5].

$\mathcal{A} = \{a_i\}_{i=0}^m \subset \mathbb{R}$, the Lagrange polynomial $\ell_j^{\mathcal{A}}(x)$ corresponding to a_j is

$$\ell_j^{\mathcal{A}}(x) = \prod_{i \neq j} \frac{x - a_i}{a_j - a_i}.$$

This has the property that $\ell_j^{\mathcal{A}}(a_k)$ is 1 if $j = k$, and 0 otherwise. Then fixing meshes $\mathcal{A} \subset \mathbb{R}$, $\mathcal{B} \subset \mathbb{R}_{>0}$, and coordinates i' and j', the template function is

$$f(x, y) = h(x, y) \cdot |\ell_{i'}^{\mathcal{A}}(x) \ell_{j'}^{\mathcal{B}}(y)|$$

where h is a hill function forcing the resulting polynomial to have compact support inside a designated box. In practice, the box for h is a bounding box containing the mesh $\mathcal{A} \times \mathcal{B}$ where both meshes $\mathcal{A}$ and $\mathcal{B}$ are chosen to have d elements; if this box further encloses all points in all diagrams, then its existence is implicit and need not be coded at all. Examples of these interpolating polynomials are shown in Fig. 4.2.

4.2 Adaptive Template Functions

We develop a modification to the template function featurization method [28]. To test this method, we apply it to synthetic shape data as well as time series data generated from the Rössler dynamical system. Here we will describe our modifications to the method as well as the results in applications.

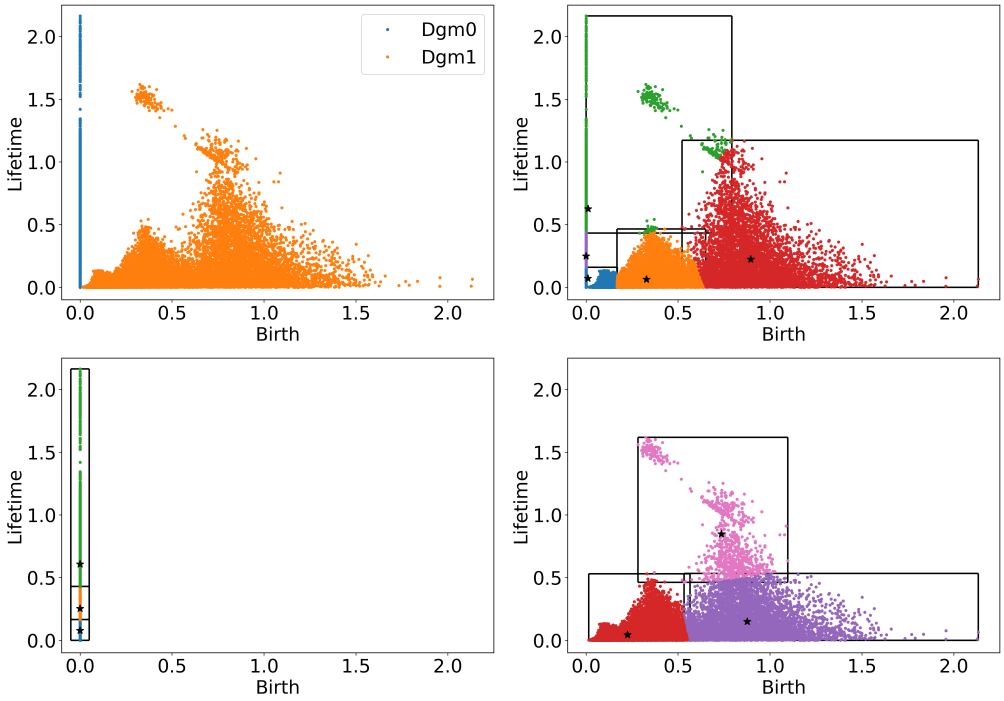


Figure 4.3: The top left image is an example of a set of persistence diagrams from the manifold experiment explained in 4.3 showing both the 0 and 1 dimensional diagrams in the birth-lifetime plane. The top right is an example showing clustering on both 0 and 1 dimensional diagrams together, which we call "combined partitioning," and creating 5 partitions. The bottom left and bottom right are examples showing 0 and 1 dimensional diagrams respectively, and clustering each dimension separately, which we call "split partitioning," creating 3 partitions per dimension. In all except the first image, the black stars represent centers of clusters from k-means clustering while the black boxes represent the partitions.

In the original template function method, a subset of a template system is selected based on a grid over the persistence diagram. However, persistence diagrams often do not have points covering the entire area of the grid. For example, in the top left plot in Fig. 4.3, the points are concentrated in certain regions of the diagram. Thus, we develop our method to select the subset of a template system based on localized information in the diagrams.

We provide an adaptive method for choosing a subset of a template system based on k-means clustering [84]. The method consists of two steps: first, cluster the points in all diagrams in a training set to find regions of interest, and second, construct localized template function systems based on these clusters.

To get the clusters, points in the persistence diagrams in the training set are combined and input

into the standard k-means clustering algorithm for a selected number of clusters k. Then, for each cluster, a covering box, which we call a partition[2], is selected based on the bounding box of the points assigned to that particular cluster. This results in one cover element per cluster; however, notice that the partitions themselves can overlap each other so points from the diagrams could land in the support of more than one partition. Because of this, the clusters themselves are not particularly interesting, they are just used to select general regions where persistence points are located. This method gives us a collection of partitions, each of which is a rectangular region in the birth-lifetime plane. We then define a grid of template functions on each partition, creating a collection of template functions for each partition. Additionally, we develop a method of adaptively selecting parameters for the template functions to fit the localized partitions.

4.2.1 Adaptive Parameter Selection

We start by describing this process for the tent functions, which have parameters d, δ, and ϵ. We develop a method of adaptively selecting d and δ based on each partition, allowing for a more localized featurization. In our modified version of the method, d does not need to be the same in the x and y direction, thus we will write d_x, d_y to specify the d parameter in each. Given a particular partition, $P = [x_{min}, x_{max}] \times [y_{min}, y_{max}]$, we first choose an initial value of parameter d. From this, δ is calculated to be $\max\{\delta_x, \delta_y\}$ where $\delta_x = \frac{x_{max} - x_{min}}{d}$ and δ_y is defined similarly. If $\delta_x > \delta_y$, then $d_x = d$ and $d_y = \lceil \frac{y_{max} - y_{min}}{\delta} \rceil$. Similarly, if $\delta_x < \delta_y$ then $d_x = \lceil \frac{x_{max} - x_{min}}{\delta} \rceil$ and $d_y = d$. The top row of Fig. 4.4 shows an example of this adaptive parameter selection process. In this example, we are using tent functions with $d = 2$. The leftmost image, we calculate δ_x and δ_y and choose δ to be the larger value. In the middle image, we select $d_y = 2$, calculate d_x as explained in Sec. 4.2 which yields $d_x = 1$, and apply a $(d_x + 1) \times (d_y + 1)$ grid (shown as the red points) where tent functions will be centered. In the rightmost image, for the tent centers that lie along the bottom of the partition (shown as a hollow blue square and solid green square), we check that the supports (shown as dashed and dotted boxes colored corresponding to their center) remain above the x-axis.

[2]Note that this is not a partition in the mathematical sense as the covering boxes can overlap.

Since they do, no further action is needed.

Note that by virtue of this notation, the support of the tent functions placed on the boundary of the partition extends outside the box. This results in a grid of size $(d_x+1)\times(d_y+1)$ which reduces the number of features used per cover element yet ensures that based on the selected d value that δ is selected appropriately to cover all points.

Additional precautions are taken to ensure that the support of the tent functions did not cross the x-axis (or the diagonal in the birth-death plane). Fix $\epsilon > 0$, a parameter chosen by the user, then if after this parameter selection $y_{min} - \delta < 0$, the grid of tent centers is shifted up to ensure the support of all tent functions is at least ϵ above the x-axis. If in this shift, there are tent centers that are greater than $\delta/2$ above the partition boundary, then they are removed and d_y is reduced by 1. The bottom row of Fig. 4.4 shows a visual example of this special case. In this example, we are using tent functions with $d = 2$. The leftmost image, we apply the same process as in the standard case but the tent supports cross the x-axis. In the middle image, we shift up the grid where tent centers are placed so the tent support is at least a small $\epsilon > 0$ above the x-axis. In the rightmost image, since the top two tent centers are more than $\delta/2$ outside the partition, we remove them, decreasing d_y by 1.

When using the interpolating polynomials, the same process as above is used to select d_x and d_y. We do not need a value of δ because the mesh is defined by a non-uniform Chebyshev mesh rather than using a regular grid like is done with the tent functions. Note that for the tent functions, we allow d_x and d_y to be zero, resulting in a grid consisting of a single, row or column of tent centers, however for the interpolating polynomials we require at least a 2×2 grid.

Note that for applications using several dimensions of diagrams, for example 0 and 1 dimensional diagrams, there are two possible options for clustering. The first is combining all diagrams in the training set regardless of dimension; the second is to combine only training persistence diagrams of like dimensions, and get a different set of clusters for each diagram dimension. Figure 4.3 shows an example of a persistence diagram with both 0- and 1-dimensional persistence in the birth-lifetime plane along with examples showing the two different methods for generating clusters when using

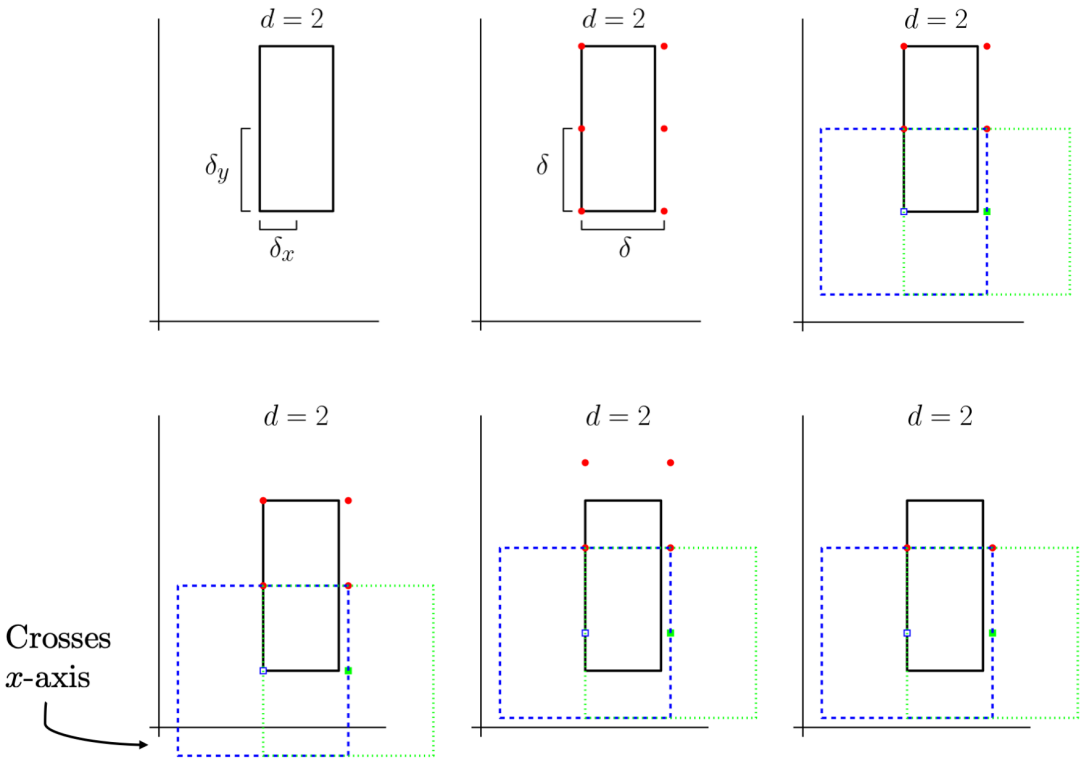


Figure 4.4: Example of steps in adaptive parameter selection for a given partition, shown as the black rectangle. The top row shows an example where the partition is far enough from the x-axis. The bottom row shows the necessary modification when the partition is too close to the x-axis.

both 0 and 1 dimensional diagrams. For simplicity, we will label results using the first option as "combined partitioning" while we will label results using the second option as "split partitioning."

4.3 Results

Here we present two applications comparing the results of the original template function method with our adaptive version. The first data set presented in Sec. 4.3.1 is a simple, proof-of-concept experiment to ensure our adaptive method is able to classify point cloud data drawn from manifolds which should be distinguishable using their topological structure. The second data set presented in Sec. 4.3.2 is a common, but fairly challenging, benchmark data set used to test persistence diagram featurization methods. The second data set presented in Sec. 4.3.3 is a collection of time series generated from the Rössler dynamical system. In this data set, the goal is to classify between

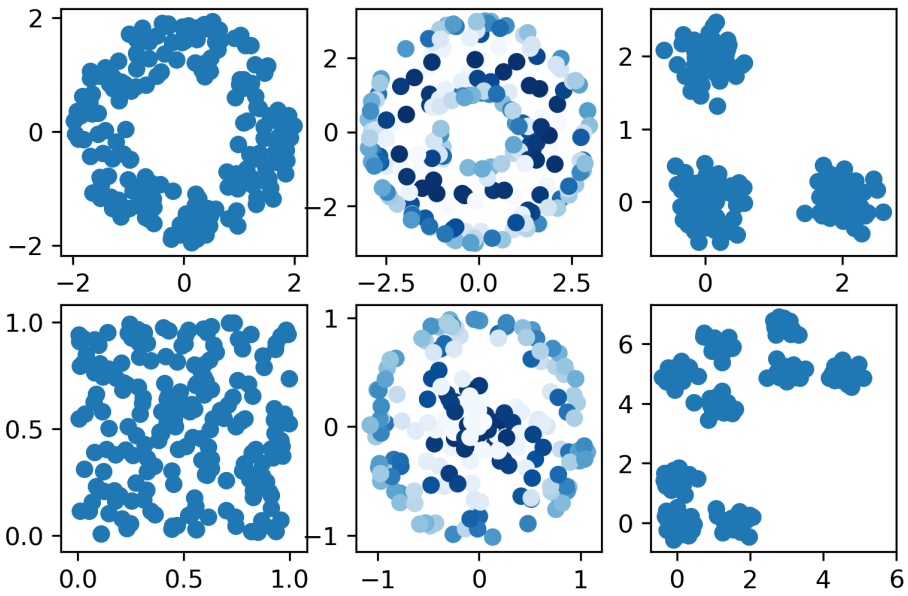


Figure 4.5: An example of each of the six types of point clouds generated for the manifold experiment. From top left to bottom right: annulus, torus, 3 clusters, square, sphere, 3 clusters of 3 clusters. In the torus and sphere, color is used to represent the third dimension.

periodic and chaotic samples.

All experiments are done using the `teaspoon` python package which has an implementation of both the original and modified template function featurization method [85].

4.3.1 Manifold Experiment

Replicating an experiment from [1, 73], we generated collections of point clouds drawn from different manifolds. Each point cloud consists of 200 points drawn from the following manifolds[3]:

- **Annulus:** points drawn uniformly from an annulus with inner radius 1 and outer radius 2.
- **Torus:** points drawn uniformly from a torus created from a rotating circle of radius 1 in the xz-plane centered at $(2, 0)$ around the z-axis.
- **Sphere:** points drawn from a sphere in $\mathbb{R}^3$ of radius 1. Uniform noise in $[-0.05, 0.05]$ was added to the radius.

[3]These point clouds can be generated using the function `MakeData.PointCloud.testSetManifolds` in `teaspoon` (https://github.com/lizliz/teaspoon)

Num Dgms	Tents			
	No Partitioning		Partitioning	
	Train	Test	Train	Test
10	99.8% ± 0.9	96.5% ± 3.2	100% ± 0.0	99.5% ± 1.5
25	99.9% ± 0.3	99.0% ± 1.0	99.9% ± 0.3	99.6% ± 0.8
50	99.9% ± 0.2	99.9% ± 0.3	100% ± 0.0	100% ± 0.0
100	99.8% ± 0.1	99.7% ± 0.4	99.9% ± 0.1	99.8% ± 0.2
200	99.5% ± 0.1	99.5% ± 0.3	99.6% ± 0.1	99.2% ± 0.3

Num Dgms	Polynomials			
	No Partitioning		Partitioning	
	Train	Test	Train	Test
10	99.8% ± 0.9	95.0% ± 3.9	100% ± 0.0	97.5% ± 2.5
25	99.7% ± 0.5	97.6% ± 1.5	99.7% ± 0.5	99.4% ± 0.9
50	100% ± 0.0	99.2% ± 0.9	100% ± 0.1	99.5% ± 0.5
100	99.6% ± 0.2	99.3% ± 0.5	99.7% ± 0.2	99.6% ± 0.5
200	99.2% ± 0.2	98.9% ± 0.5	99.5% ± 0.2	99.4% ± 0.3

Table 4.1: Results of classification of manifold data using template functions with and without partitioning for different numbers of examples drawn from each type of manifold. Ridge regression is used for classification in both methods. Scores highlighted in green give the best average score between the two methods.

- **Square:** points drawn uniformly from $[0, 1]^2 \subset \mathbb{R}^2$.
- **3 Clusters:** points drawn from one of three different normal distributions with means $(0, 0)$, $(0, 2)$, $(2, 0)$, each with standard deviation of 0.05.
- **3 Clusters of 3 Clusters:** points drawn from normal distributions centered at (0,0), (0,1.5), (1.5,0), (0,4), (1,3), (1,5), (3,4), (3,5.5), (4.5,5) each with standard deviation 0.05.

Figure 4.5 shows an example of each of these manifolds.

For our method we tested a variety of parameters and options. For all experiments, we reserve 33% of the data for testing while the remaining was used for training. All the classification results are averaged over 10 runs of the experiment to control for outliers.

For comparison, Table 4.1 shows the results from [1] using 0- and 1-dimensional diagrams with ridge regression for classification along with our accuracies using our partitioning method where partitions are selected based on both diagram dimensions simultaneously, referred to as "combined partitioning." For the results from [1], the authors used tent parameters of $d = 10$, ϵ to be half the

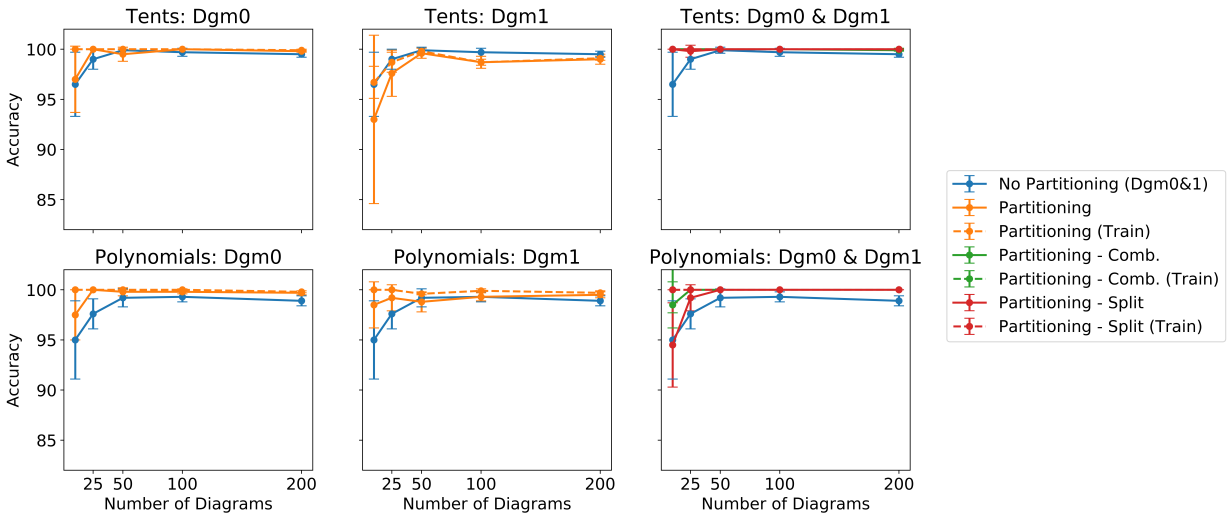


Figure 4.6: Results of classification of manifold data using template functions with and without partitioning. For partitioning methods, classification is done using logistic regression. Note that in all plots, the results without partitioning represent the accuracy using 0- and 1-dimensional diagrams. Thus the same accuracy is shown in each plot in a row.

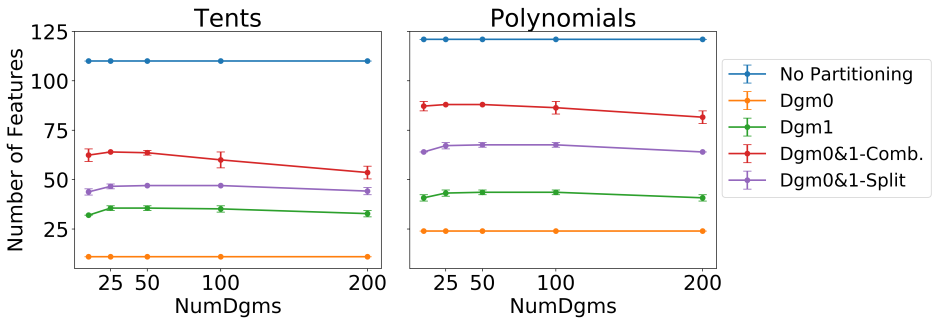


Figure 4.7: Average number of features used for the manifold experiment using template functions with and without partitioning. These correspond to the classification accuracies shown in Fig. 4.6.

minimum lifetime over all training set diagrams, and δ to be chosen to ensure the bounding box covered the training diagrams. For our results, we used 3 clusters resulting in 3 partitions and for both template functions we set a starting value of $d = 3$ and set ϵ to be machine precision, while the additional parameters are selected as described in Sec. 4.2.1. Note that in all cases except one, our partitioning method has a higher testing accuracy.

Figure 4.6 shows the results of using our partitioning method on only 0- or 1-dimensional diagrams; on both dimensions using combined partitioning; and on both dimensions using split partitioning. Here we still used 3 clusters resulting in 3 partitions, while starting with a value

of $d = 3$ and for both template functions. For these experiments we used logistic regression for classification. It is important to note that in [1], only accuracies using both dimensions were reported so the accuracies for no partitioning in all plots are from using both dimensions of diagrams with classification done using ridge regression. This means that those results are the same across all plots for a given template function.

Using both 0- and 1-dimensional diagrams with either split or combined partitioning, using both tent functions and interpolating polynomials we get above 99% accuracy for most cases. Using only 0-dimensional or only 1-dimensional diagrams, we still get very good accuracy, but interestingly using 0-dimensional diagrams, the accuracies seem slightly better. Using only 0-dimensional diagrams, we almost always outperform the template functions without partitioning using both 0- and 1-dimensional diagrams. However, using tent functions with 1-dimensional diagrams, our method under-performs. Additionally, Fig. 4.7 shows the average number of features used for these experiments. The number of features used is dependent on which diagrams are being used. For example, using only 0-dimensional diagrams, we need very few features as all points in the diagrams fall on the y-axis for this experiment. Using both 0- and 1-dimensional diagrams will require more features as we need to cover more of the diagram. However in all cases, we are using significantly less features and still achieving comparable or higher accuracies.

4.3.2 Shape Dataset

Here we present another application of the adaptive template function method and compares the results to the original template functions in [1] as well as the persistence diagram kernel method developed in [2]. This data set, called the synthetic SHREC 2014 data set [86], consists of 3D meshes of fifteen humans (five males, five females and five children) in 20 different poses, resulting in 300 meshes in total. In [2], the authors define a function on each mesh using the heat kernel signature for 10 different parameters and compute 0- and 1-dimensional diagrams. Using the 300 pairs of persistence diagrams for each 10 parameter values, we predict which of the 15 humans is represented in each mesh.

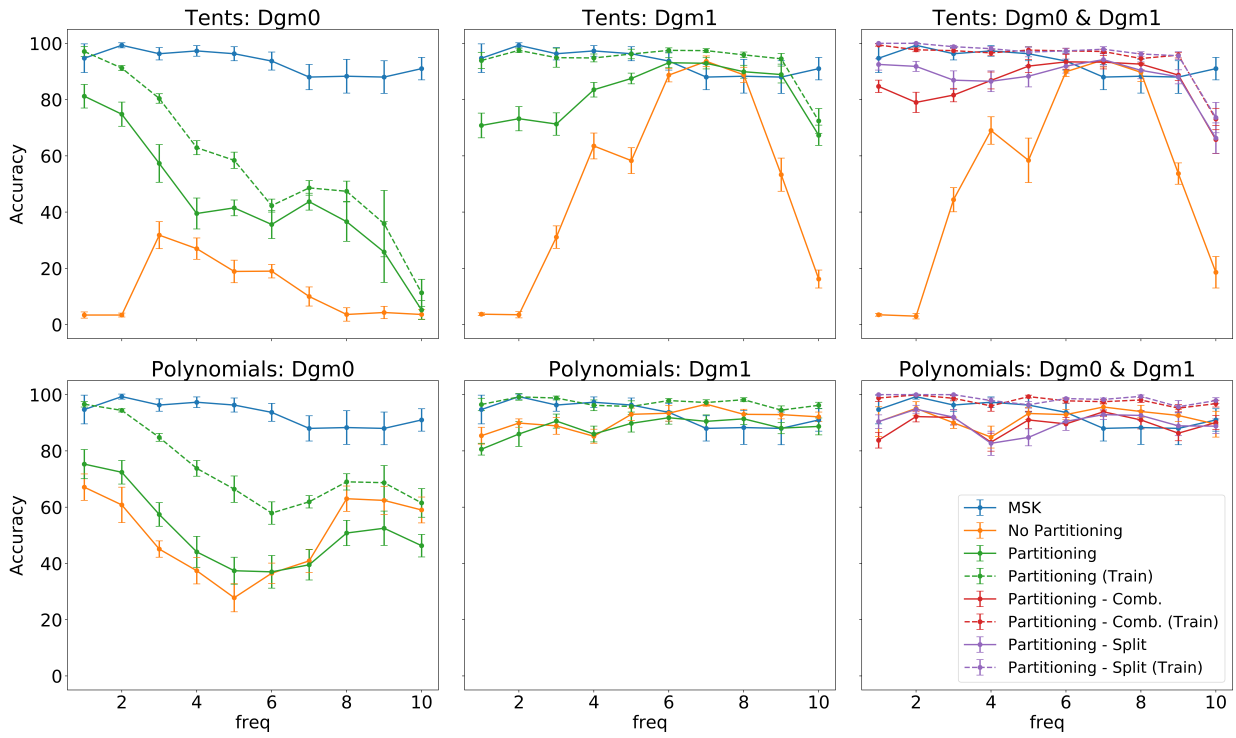


Figure 4.8: Results of classification of shape data using template functions with and without partitioning. MSK gives the original results from [2]. Ridge regression is used for classification.

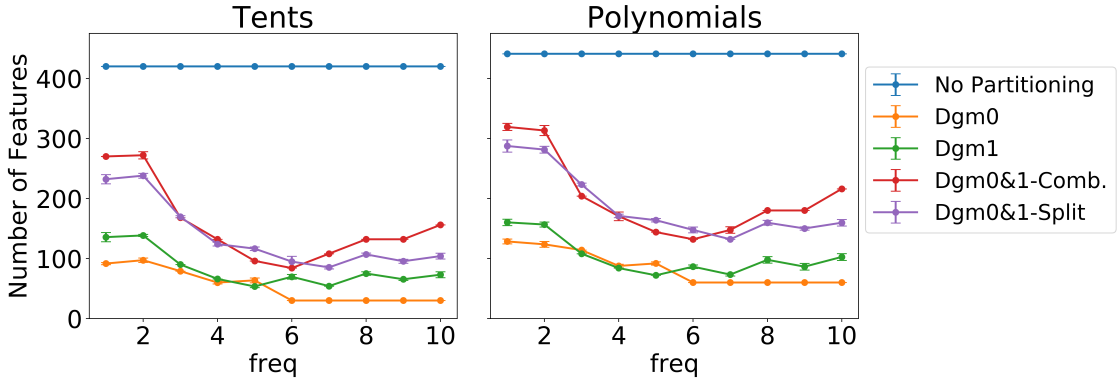


Figure 4.9: Average number of features used for shape data experiment using our partitioning method. These correspond to the classification accuracies shown in Fig. 4.8.

Figure 4.8 show the results of these experiments using our partitioning method with tent and interpolating polynomial functions as well as the results reported in [1] (labeled as "No Partitioning") and [2] (labeled as "MSK"). For clarity, tables showing these results are located in Appendix A in Tables A.1 and A.2. For all experiments with partitioning, we use $d = 5$ and 5 clusters for partitioning. For the experiments without partitioning, the authors use $d = 20$ for both

types of template functions. All accuracies with and without partitioning are averaged over 10 runs.

For three of the ten parameter values, using tent functions, our method achieves the highest accuracy. Additionally, the confidence intervals intersect the highest accuracy for three additional parameters using tent functions. Comparing tent functions with and without partitioning, it is clear that partitioning drastically improves most of the accuracies. For example, using 0- or 1-dimensional diagrams, the top left and middle left plot in Fig. 4.8, the green line representing our testing accuracy is almost always higher than the orange line representing the testing accuracy without partitioning.

Using interpolating polynomials, our method does not surpass the kernel method or the template functions without partitioning to achieve the highest accuracy, however the confidence intervals intersect the highest accuracy for seven of the ten parameters. Comparing our results to featurization using interpolating polynomial functions without partitioning, our results are fairly comparable; for some parameter values we achieve slightly higher accuracies, while for others we achieve slightly lower accuracies. Without partitioning, the interpolating polynomials are not localized and may be picking up more global structure in the diagrams that is missed using partitioning, which could explain this lack of improvement.

Figure 4.9 shows the average number of features used for these experiments. It is clear that we are using far less than the 420 and 441 features used in [1] with tent and polynomial functions respectively, yet we still achieve good accuracies, particularly using tent functions. For example, for the parameters where tent functions with partitioning achieve the highest accuracy, we use less than 150 features.

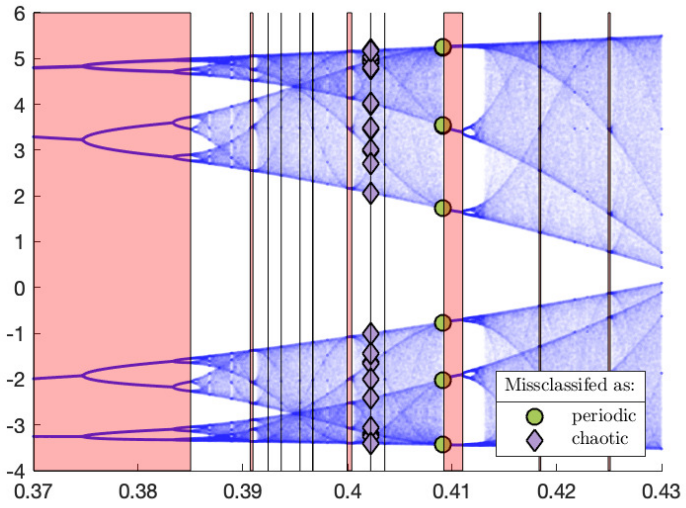


Figure 4.10: The bifurcation diagram for the Rössler system with α as the bifurcation parameter. The shaded windows indicate the regions that were tagged as periodic. The misclassified points are superimposed with diamonds indicating the points that were incorrectly identified as chaotic, while dots indicate that the algorithm incorrectly identified chaotic points as periodic.

4.3.3 Rössler System

We also test this method on time series simulated from the Rössler system [87]:

$$\begin{aligned} \frac{dx}{dt} &= -y - z, \\ \frac{dy}{dt} &= x + \alpha y, \\ \frac{dz}{dt} &= \beta + z(x - \gamma). \end{aligned} \tag{4.2}$$

The Rössler system is solved for $\beta = 2, \gamma = 4$, and 1201 evenly spaced values of the bifurcation parameter α, with $0.37 \leq \alpha \leq 0.43$. For each of these parameter values, 2×10^4 points are sampled using a time step of 0.2 seconds, and the first half of the points are discarded.

As described in [1, Sec.8.5], when the original template function featurization method is used with tent functions, they get a training score of 98.9% and a testing score of 97.2%. They use $d = 10$, which corresponds to 110 features. We test the adaptive method with tent functions using 3 partitions and $d = 3$. Using random forest classifier, we achieve a training accuracy of 100% and testing accuracy of 99.5% only using dimension 1 diagrams. This means we only misclassify two values of α, Specifically, $\alpha = 0.40220$ is tagged as periodic, however our method classifies it as

chaotic, while $\alpha = 0.40915$ is tagged as chaotic, while our method classifies it as periodic. Note that both of these values of α occur on transitions of the system behavior from periodic to chaotic or vice versa. It is worth noting that the ground truth labels in this experiment are determined by the Lyapunov exponent [88] which is a tool from dynamical systems that is often used to classify periodic and chaotic behavior. While accurate, the Lyapunov exponent is not a perfect ground truth, and thus it is possible that a misclassification is actually detecting an incorrect label determined by the Lyapunov exponent.

For this classification, our method only uses 33 features. Therefore compared to the original template function method, we can use less than a third the number of features while still achieving slightly better accuracy. Overall, both the original template function featurization and our adaptive version can successfully classify periodic versus chaotic behavior in this dynamical system.

CHAPTER 5

TIME SERIES CLASSIFICATION VIA PERSISTENT HOMOLOGY OF DIRECTED NETWORKS

The time delay embedding as described in Sec. 2.3 is a useful tool but the number of points in the point cloud grows with the length of the time series, thus increasing the computation time for persistent homology. In this chapter, we utilize an alternative method to instead transform a time series into a network, which provides a coarser summary of the data but still captures important features.

5.1 Basic Graph Definitions

A graph[1], $G = (V, E)$ is a collection of vertices, V, with edges, $E = \{uv\} \subseteq V \times V$. Graphs can also be directed where the edges have a direction. An example of an undirected and a directed graph can be seen in Fig. 5.1. In general, we will denote an undirected edge between $u, v \in V$ as uv, and a directed edge from u to v as $\overrightarrow{uv}$. Here we will allow self-loops, specifically edges of the form $\overrightarrow{vv}$, as well as multiedges going in opposite directions, i.e. $\overrightarrow{uv}, \overrightarrow{vu} \in E$. For the remainder of this section, the definitions are the same in the directed and undirected case.

The complete graph on the vertex set $V = \{v_1, \ldots, v_n\}$ is the graph with edge set $E = \{v_i v_j \text{ for all } i, j\}$. We also consider weighted graphs, $G = (V, E, \omega)$, where $\omega : E \to \mathbb{R}^+$ specifies the weight of each edge. For the remainder of this section we will give all definitions in terms of weighted graphs since unweighted graphs can be thought of as weighted graphs where all edges have a weight of 1. A graph can be represented as an adjacency matrix $\mathbf{A}$, where, given an ordering on the vertex set, $V = \{v_0, \ldots, v_n\}$,

$$\mathbf{A}[i, j] = \begin{cases} \omega(v_i v_j) & \text{if } v_i v_j \in E \\ 0 & \text{otherwise.} \end{cases}$$

[1]Note that throughout this chapter we will often use the terms *network* and *graph* interchangeably.

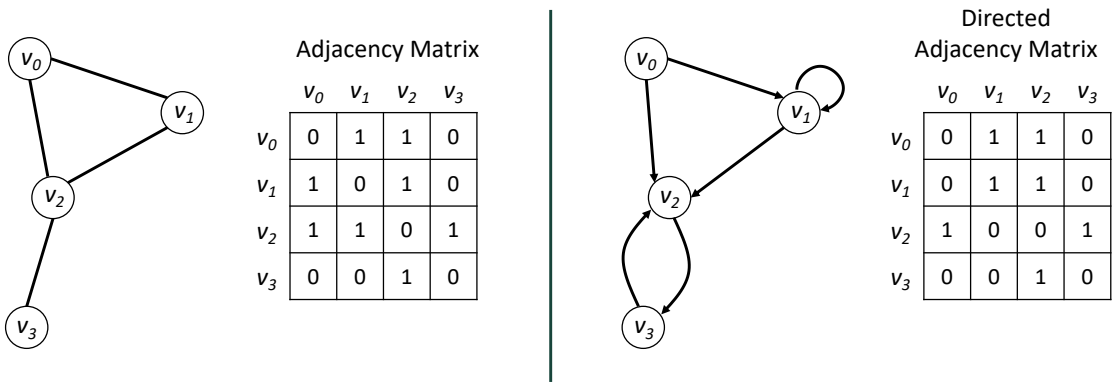


Figure 5.1: Example of undirected (left) and directed (right) graphs with corresponding adjacency matrices.

Note that for an undirected graph, the adjacency matrix is symmetric however for a directed graph it is not. This is due to the fact that $\overrightarrow{v_i v_j}$ and $\overrightarrow{v_j v_i}$ are two different edges and the presence of one does not imply the presence of the other. Figure 5.1 shows the adjacency matrices corresponding to each graph.

In an undirected graph, the degree of a vertex is the number of edges connected to that vertex. To summarize this information across an entire graph, one can consider the mean and standard deviation of the degree of all vertices. In the undirected graph in Fig. 5.1, the degree of v_2 is 3, the degree of v_0 and v_1 is 2 and the degree of v_3 is 1. In the case of a directed graph, instead one can calculate the indegree and outdegree, counting the number of edges going into and out of the vertex, respectively. In the directed graph in Fig. 5.1, the indegree of v_2 is 3, while the outdegree is 1. Again, one can summarize the information in these networks with the mean and standard deviation of the in or outdegrees of all vertices in the network.

A path γ on a graph $G = (V, E)$ is a sequence of vertices, $u_0 u_1 \dots u_n$, where $u_i \in V$ and $u_i u_{i+1} \in E$ for all i. In the case of directed graphs, the edges must be followed along the direction of the edge. For example, in the directed graph in Fig. 5.1, $v_0 v_1 v_2$ is a path, but $v_0 v_2 v_1$ is not since $\overrightarrow{v_2 v_1} \notin E$. The length of a path γ, $\text{len}(\gamma)$ is the sum of the weights of the edges used in the path,

$$\text{len}(\gamma) = \sum_{i=0}^{n} \omega(u_i u_{i+1}).$$

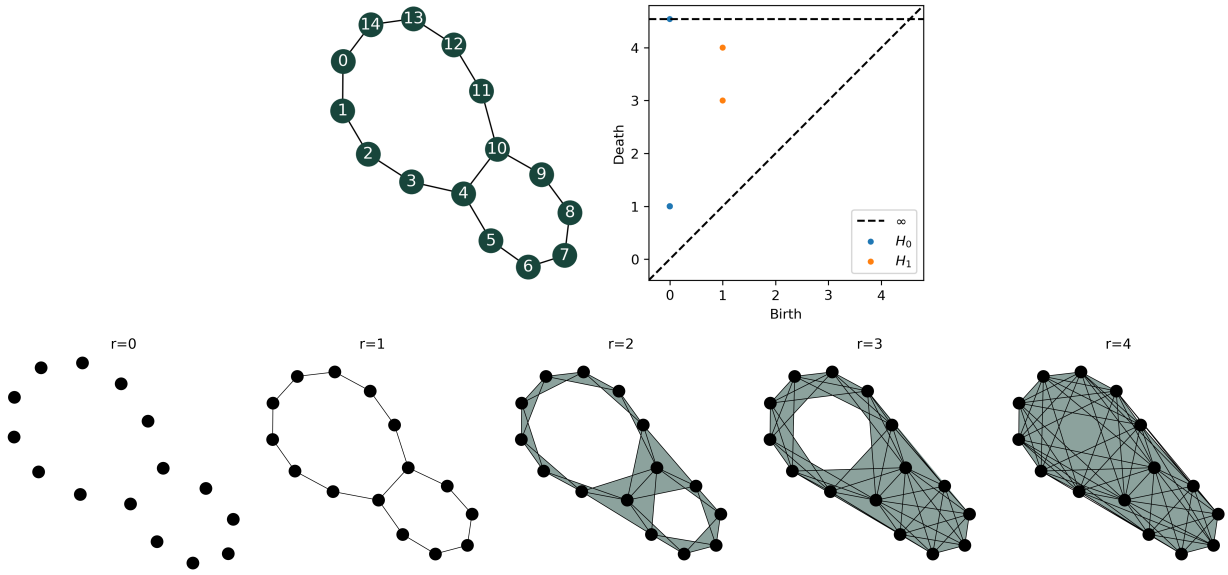


Figure 5.2: Example of a graph with its corresponding persistence diagrams, along with several steps in the filtration generated from the clique complexes. In this example we assume each edge has a weight of 1.

From this, we can define the geodesic distance between vertices $u, v \in V$ in the graph,

$$d(u, v) = \min_{\gamma \in \Gamma_{u,v}} \text{len}(\gamma),$$

where $\Gamma_{u,v}$ is the set of paths that start at u and end at v. In an undirected graph, this distance is symmetric (i.e. $d(u, v) = d(v, u)$) but not in the directed case. Given a (directed) weighted graph, $G = (V, E, \omega)$ we can compute a (directed) weighted complete graph on V where $\omega(uv) = d(u, v)$. The adjacency matrix of this graph is then a pairwise distance matrix on all vertices in V.

5.2 Persistent Homology on Graphs

For the undirected case, we can turn a graph into a simplicial complex using clique complexes. Given a graph, $G = (V, E)$, the clique complex of G is defined as

$$\mathcal{K}(G) = \{\sigma \subset V \mid uv \in E \text{ for all } u \neq v \in \sigma\}.$$

Given a weighted graph $G = (V, E, \omega)$, we can construct a filtration of clique complexes. Given $r \in \mathbb{R}$, the clique complex at scale r is defined as

$$\mathcal{K}_r = \{\sigma \in K(G) \mid \omega(uv) \leq r \text{ for all } u \neq v \in \sigma\}.$$

Then $\mathcal{K}_{r_0} \subseteq \mathcal{K}_{r_1}$ for $r_0 \leq r_1$, thus we can construct a filtration

$$\mathcal{K}_{r_0} \subseteq \mathcal{K}_{r_1} \subseteq \ldots \subseteq \mathcal{K}_{r_n}$$

for $r_0 \leq r_1 \leq \cdots \leq r_n$.

As described in Sec. 5.1, we can create a pairwise distance matrix using the shortest path distance. This can then be viewed as a weighted, complete graph, and gives rise to a filtration of clique complexes. For example, the top left in Fig. 5.2 shows a graph where we will assume every edge has weight 1. The bottom row of Fig. 5.2 shows several steps in the filtration using various values of threshold r. For each value of r, you add an edge between every pair of vertices such that there exists a path γ between the vertices with $\text{len}(\gamma) \leq r$. Then a triangle is added when all its pairwise edges are added. In this example, at $r = 0$, all the vertices in the network are included, then at $r = 1$, all the edges in the graph are then added. At $r = 2$ we start seeing triangles included since there are now sets of three vertices with all pairwise edges included. The smaller loop fills in at $r = 3$ while the larger loop fills in at $r = 4$; this is reflected in the points $(1, 3)$ and $(1, 4)$ in the 1-dimensional persistence diagram also shown in Fig. 5.2. In the case of undirected graphs where all edges have weight 1, all 1-dimensional features are born at 1, and all death times are integer valued.

Note that this section focused entirely on undirected graphs. We will address incorporating directed information in Sec. 5.4.2.

5.3 Ordinal Partition Graph

There are many methods to transform a time series into a graph [89–94] and a survey of many of them is presented in [95]. Here we will focus on one of these methods called the ordinal partition graph (also called the ordinal partition network) [87, 96].

To generate the ordinal partition graph, one must first apply the delay embedding method described in Sec. 2.3. Then for each $\mathbf{x}_i = (x_1, \ldots, x_d) \in \mathbf{X}$, the permutation π of the set $\{1, \ldots, d\}$ satisfying $x_{\pi(1)} \leq x_{\pi(2)} \leq \cdots \leq x_{\pi(d)}$ is associated to $\mathbf{x}_i$. Thus, each vector is translated into its associated permutation π_j to generate a collection of permutations where $j \in \mathbb{Z} \cap [1, n!]$. The

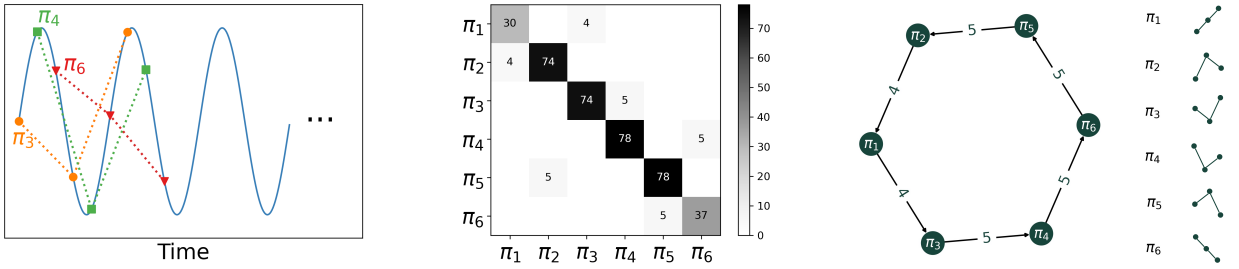


Figure 5.3: Example ordinal partition network. (a) Time series showing three permutations; (b) Adjacency matrix generated from time series; (c) Directed, weighted ordinal partition network.

permutations become the vertices in the graph, and an edge $\pi\pi'$ is added whenever the permutations associated to the ordered point cloud passes between π and π'. This can also be modified to be a directed graph, where the edges point in the direction of the transition, i.e. $\overrightarrow{\pi\pi'}$. Additionally, the graph can be weighted by setting weight $\omega(\pi, \pi')$ to be the number of times the transition occurs.

One benefit of this method over the time delay embedding is that an upper bound on the number of vertices in the network is determined by the chosen embedding dimension. While in theory, an ordinal partition network with dimension d could have $d!$ vertices, in practice we do not include vertices associated to permutations that are not visited.

These networks have shown success in experiments on dynamical systems such as the Rössler system as well as experimental data from ECG data [97] and a diode resonator circuit [87]. Statistics such as the number of vertices or mean and standard deviation of the vertex degrees (or in the directed case, in or out degree) were found to be useful in characterizing the networks [87, 96]. There is also a lot of work going into exploring this method further. For example, [98] presents an approach to generate ordinal partition networks from multivariate time series data, while [99] explores whether a time series can be reconstructed from a given ordinal partition network.

5.3.1 Parameter Selection

As with the time delay embedding, the ordinal partition graphs require two parameters: the delay τ and the dimension d. In this chapter we describe one heuristic based on multi-scale permutation entropy (MsPE) to select the delay [43].

Permutation entropy is a tool to measure complexity in a time series [100]. It is defined on a time series based on permutations as defined in Sec. 5.3. Given a time series, one can calculate the probability of a permutation $P(\pi_i)$ by calculating at how many times permutation π_i occurs divided by the total number of permutations[2]. Using an embedding dimension of n, then permutation entropy is defined as

$$H(n) = -\sum_i P(\pi_i) \log_2 P(\pi_i). \tag{5.1}$$

Further, it can be normalized by the maximum possible permutation entropy value which occurs when all permutations are equally probable. The normalized permutation entropy is defined as

$$h(n) = -\frac{1}{\log_2 n!} \sum_i P(\pi_i) \log_2 P(\pi_i). \tag{5.2}$$

The process to choose τ based on normalized permutation entropy is iterative. For each possible τ value, normalized permutation entropy can be recalculated, then the ideal value of τ should be chosen to be the first peak in the normalized permutation entropy. An algorithm to calculate τ based on permutation entropy can be found in [43] and is implemented in `teaspoon` [85].

5.3.2 Examples and Existing Work

Figure 5.4 show examples of the ordinal partition networks from a periodic and chaotic time series from both the Rössler and Lorenz systems. In both cases, the embedding dimension is selected to be $d = 6$ while the delay is selected for each time series individually using MsPE. These networks look visually different when they are constructed from periodic or chaotic time series. Immediately it is noticeable that the topological structure, specifically the number of loops and the size of the loops, is different.

In [101], the authors use the topological structure of unweighted, undirected ordinal partition networks to differentiate between chaotic and periodic behavior. They also showed that using topological features were more effective than the previously used network based statistics such as

[2]Note we mean the total number of permutations including duplicates.

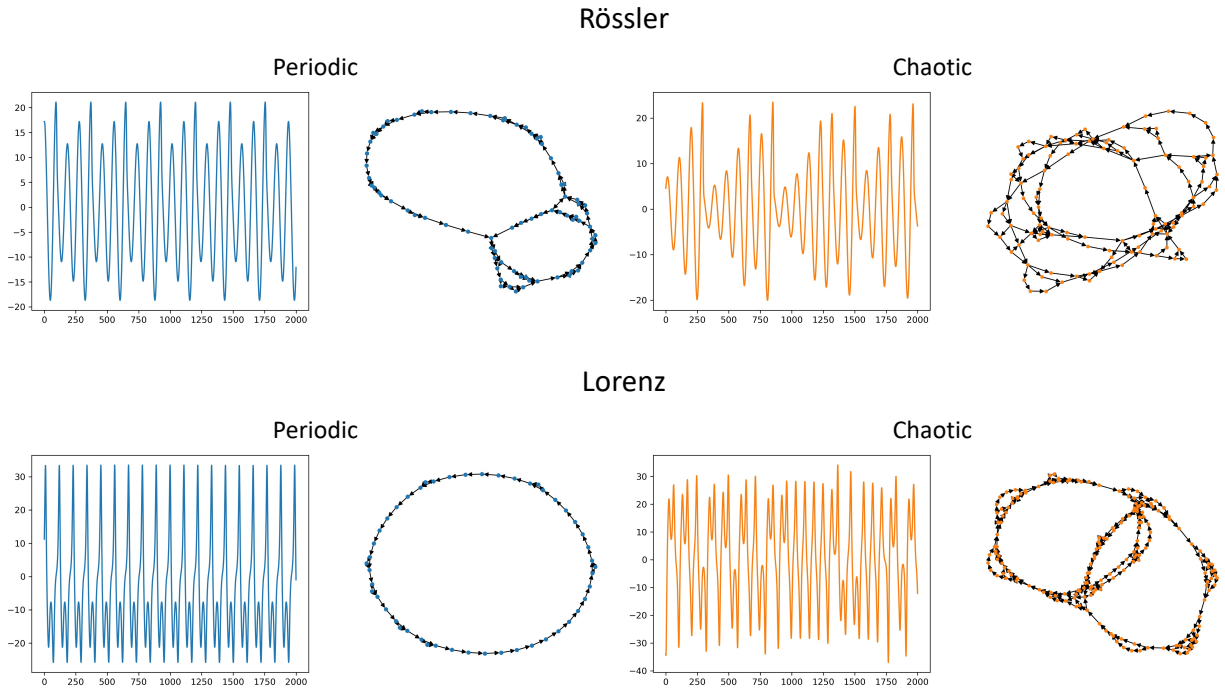


Figure 5.4: Examples of time series and the corresponding ordinal partition networks. Top row is from the Rössler system, bottom row is from the Lorenz system.

mean degree or number of vertices. Their method is successful on several examples from dynamical systems however the method does not perform as well when the time series have added noise.

5.4 Directed, Weighted Ordinal Partition Networks

Here we explore incorporating the weighted and directed information from the ordinal partition networks to test if it can improve upon the results in [101]. In order to explore the topological structure of weighted, directed ordinal partition networks, we need to take three steps. First, we need to determine how to use the weights. We cannot use the weights as calculated because a transition that occurs frequently will have a larger weight than a transition that occurs only a few times. Because of the way we construct the filtration, we would like more highly used (thus more important) edges to have a lower filtration value. Additionally, the weights on the edges relate closely to the length and sampling rate of the time series, but these factors should not be incorporated in the analysis. Thus the raw counts of the number of times a transition occurs is not a useful weighting scheme, so we will instead calculate a new set of weights using these counts.

Second, once we have a better weighting scheme, we need a way to compute persistent homology of directed, weighted graphs. Thus far, we have not discussed any filtrations or methods to compute persistent homology that would apply in the case of directed graphs. Lastly, we need to analyze the resulting persistence diagrams. In [101], the authors use scores or statistics[3] on persistence diagrams, where you compute one number summarizing some feature of the diagram. We will test three different statistics which will be introduced in Sec. 5.4.3.

Code implementing these methods are included in the `teaspoon` python library.

5.4.1 Step 1: Computing the weights

As mentioned, there are many flaws with using the raw counts between transitions, so instead we use these counts to calculate a new set of weights based on inverse probability. For example, the weight of an edge $\overrightarrow{\pi_i \pi_j}$ is 1 minus the probability that you transition to permutation π_j given that you are standing at π_i. This can be computed from the adjacency matrix A with the directed transition counts as

$$\widetilde{\omega}(\pi_i, \pi_j) = 1 - \frac{A[i,j]}{\sum_k A[i,k]}. \tag{5.3}$$

Note we use 1 minus the probability because we want edges used more often to have a lower weight, thus appearing earlier in a sublevel set filtration. Note that as can be seen in Fig. 5.3, we also keep track of the counts of how many steps the same permutation occurs, recorded on the diagonal of the adjacency matrix. However this number is related to the sampling of the time series, so instead we make a slight modification by assuming the probability of remaining in the same permutation is 50%. This makes it uniform across all permutations and ensures the spacing along the time series is not influential. Thus, we modify our weighted scheme to be

$$\omega(\pi_i, \pi_j) = \begin{cases} 0.5 - \frac{A[i,j]}{\sum_{k!=i} A[i,k]} & \text{if } i \neq j \\ 0.5 & \text{if } i = j. \end{cases} \tag{5.4}$$

[3] We will use the terms *score* and *statistic* interchangeably.

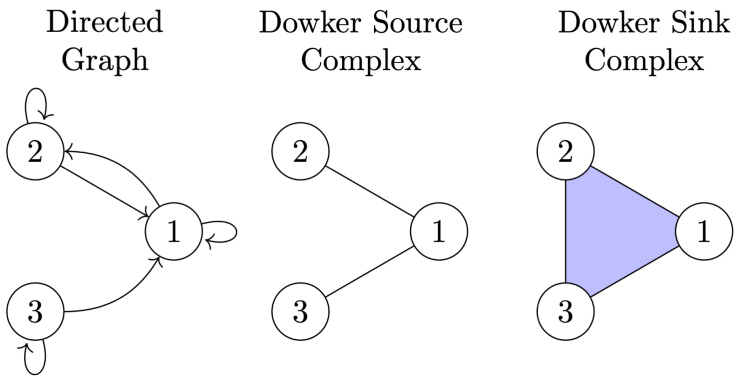


Figure 5.5: Example of a directed graph and the corresponding Dowker source and sink complexes.

Now using these weights, we can compute a shortest path distance matrix, which represents a directed, weighted, complete graph. This approach is inspired by Markov chains and graph diffusion distances [102, 103].

5.4.2 Step 2: Viewing a directed, weighted graph as a (filtered) simplicial complex

While weighted graphs naturally give rise to a filtration as described in Sec. 5.2, incorporating directed information requires some additional thought. Here we will focus on one method of transforming a directed graph into a simplicial complex called the Dowker complex [104]. Given a directed graph $G = (V, E)$ the Dowker source complex is defined as

$$\mathfrak{D}^{so}(G) = \{\sigma = (v_0, v_1, \ldots, v_n) : \exists\, v' \in V \text{ with } \overrightarrow{v'v_i} \in E \text{ for } i = 1, \ldots, n\}$$

Intuitively, this means that for a given vertex v', we add an edge between any two vertices v_i, v_j where there are edges $\overrightarrow{v'v_i} \in E$ and $\overrightarrow{v'v_j} \in E$. We add a triangle between any three vertices v_i, v_j, v_k when $\overrightarrow{v'v_i}, \overrightarrow{v'v_j}, \overrightarrow{v'v_k} \in E$, and so on for higher dimensional simplices. Similarly, the Dowker sink complex is defined as

$$\mathfrak{D}^{si}(G) = \{\sigma = (v_0, v_1, \ldots, v_n) : \exists\, v' \in V \text{ with } \overrightarrow{v_iv'} \in E \text{ for } i = 1, \ldots, n\}$$

An example of a directed graph with the corresponding Dowker source and sink complexes are shown in Fig. 5.5.

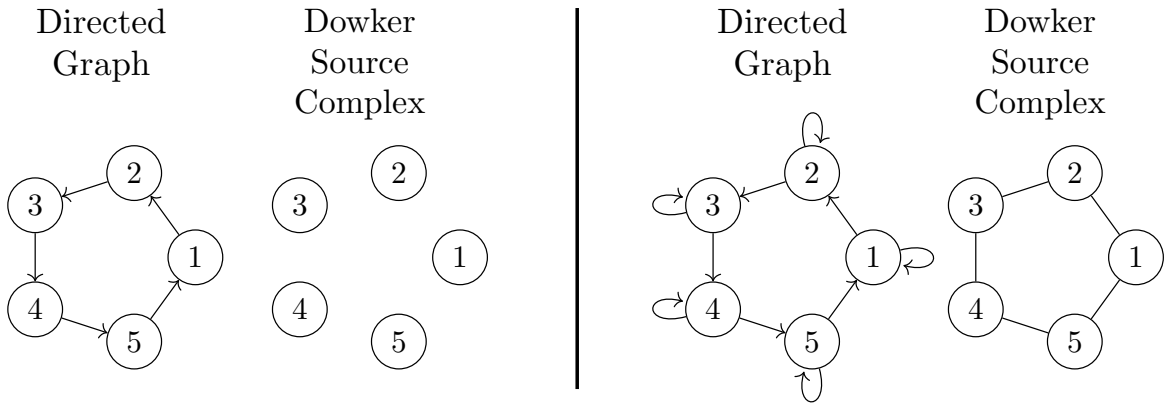


Figure 5.6: Example of a directed cycle graph without and with self-loops and the corresponding Dowker source complexes.

Similarly, given a directed, weighted graph $G = (V, E, \omega)$ we can define the Dowker source and sink complex at scale $r \in \mathbb{R}$,

$$\mathfrak{D}_r^{so}(G) = \{[v_0, v_1, \ldots, v_n] : \exists\, v' \in V \text{ with } \overrightarrow{v'v_i} \in E \text{ and } \omega(\overrightarrow{v'v_i}) \leq \delta \text{ for } i = 1, \ldots, n\}$$
$$\mathfrak{D}_r^{si}(G) = \{[v_0, v_1, \ldots, v_n] : \exists\, v' \in V \text{ with } \overrightarrow{v_iv'} \in E \text{ and } \omega(\overrightarrow{v_iv'}) \leq \delta \text{ for } i = 1, \ldots, n\}.$$

Given an increasing sequence of values $r_0 \leq r_1 \leq \cdots \leq r_n$ with $r_i \in \mathbb{R}$ for all i, we get the Dowker source and sink filtrations,

$$\mathfrak{Dow}^{so} = \mathfrak{D}_{r_0}^{so} \subset \mathfrak{D}_{r_1}^{so} \subset \cdots \subset \mathfrak{D}_{r_n}^{si} \tag{5.5}$$
$$\mathfrak{Dow}^{si} = \mathfrak{D}_{r_0}^{si} \subset \mathfrak{D}_{r_1}^{si} \subset \cdots \subset \mathfrak{D}_{r_n}^{si}. \tag{5.6}$$

While the Dowker source and sink complexes are not equal in general, it has been proven in [104] that the k-dimensional persistence diagrams of the filtrations in Eqns. 5.5 and 5.6 are equal. That is,

$$Dgm_k(\mathfrak{Dow}^{so}(G)) = Dgm_k(\mathfrak{Dow}^{si}(G)) \tag{5.7}$$

for all $k = 0, 1, 2, \ldots$. Thus, we can choose one and call it the Dowker filtration. In practice, we use the source filtration.

Note that the Dowker complex is sensitive to the presence of self-loops. For example, the Dowker complex of a directed cycle graph with no self-loops is only the disconnected vertices.

On the other hand, the Dowker complex of the same graph with self-loops has edges connecting adjacent vertices. Figure 5.6 shows an example of each case. This means that the weighting of self-loops (which corresponds to staying in the same permutation) as defined in Sec. 5.4.1 is fairly important as the lack of even one self-loop it can change the topological structure of the resulting simplicial complex.

5.4.3 Step 3: Analyzing the Persistence Diagrams

The first two steps allow us to compute a persistence diagram from a directed, weighted ordinal partition network. Specifically, we compute a persistence diagram using the Dowker filtration on the weighted, directed, complete graph built from a shortest path distance matrix.

The next step is to determine how to analyze the persistence diagrams. While methods for machine learning on persistence diagrams as described in Chapter 4 are very useful, they can be computationally expensive and require a significant amount of data, thus a simpler method is more useful in this case. For our experiments, we will calculate three different statistics on each persistence diagram: (1) total persistence, (2) maximum persistence, and (3) persistent entropy.

We define total persistence to be the sum of the lifetimes of all points in a given diagram D,

$$\text{TotalPers}(D) = \sum_{(b,d)\in D} d - b. \tag{5.8}$$

Note that this is equal to the 1-Wasserstein distance as defined in Eqn. 2.4 (using $p = q = 1$) between D and the empty diagram, $\text{TotalPers}(D) = d_{W_1}(D, \emptyset)$. Maximum persistence was already defined in Eqn. 2.6. Note that maximum persistence is also equivalent to computing twice the bottleneck distance between a diagram D and the empty diagram, $\text{MaxPers}(D) = 2d_{W_\infty}(D, \emptyset)$. Persistent entropy[4] (PE), as defined in [105], calculates the entropy in a persistence diagram. Inspired by Shannon entropy, it characterizes the differences in lifetimes of the persistence points. It is defined as

$$E(D) = -\sum_{(b,d)\in D} \frac{d-b}{\text{TotalPers}(D)} \log_2\left(\frac{d-b}{\text{TotalPers}(D)}\right).$$

[4]Not to be confused with permutation entropy from Sec. 5.3.1.

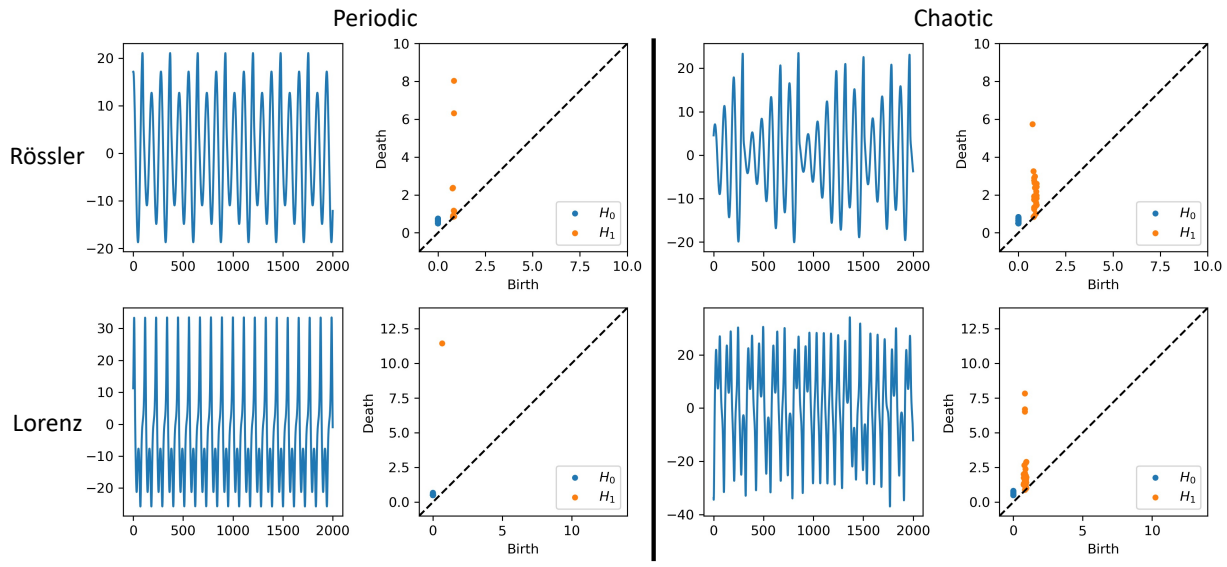


Figure 5.7: Example of periodic and chaotic time series from two different systems (Rössler and Lorenz) and the corresponding persistence diagrams.

Note that this value can vary based on the number of points in the persistence diagram, so one can normalize persistent entropy as

$$PE(D) = \frac{E(D)}{\log_2(\mathcal{L}(D))}. \tag{5.9}$$

For the remainder of this chapter, when we refer to persistent entropy, we mean this normalized version.

Persistent entropy was found to be a useful statistic on persistence diagrams generated from unweighted, undirected ordinal partition networks in [101]. However, we note that persistent entropy is not well grounded in theory as it is continuous but not stable [106]. We will compare the effectiveness of each of these three statistics across a range of preliminary experiments.

5.5 Preliminary Results

Now using the steps provided in Sec. 5.4 we have a process to convert a directed, weighted ordinal partition network into a persistence diagram. In order to test our method, we will perform several experiments to find strengths and weaknesses of the methods.

In all our experiments, we will generate data using the Dynamical Systems Library in the python `teaspoon` library [85], specifically the `DynamicSystems` function. Further, in all cases we

	TotalPers		MaxPers		PE	
System	**Periodic**	**Chaotic**	**Periodic**	**Chaotic**	**Periodic**	**Chaotic**
Rössler	16.63	34.95	7.18	4.99	0.49	0.88
Lorenz	10.77	56.84	10.77	6.99	0.00	0.88

Table 5.1: Statistics calculated on the persistence diagrams in Fig. 5.7. Scores are rounded to two decimal places.

use an embedding dimension of $d = 6$ for the ordinal partition networks, and delay τ is selected using multi-scale permutation entropy [43] as described in Sec. 2.3 implemented in `teaspoon`. All the default parameters in teaspoon are used except the `SampleSize` parameter which dictates how long the generated time series is. The function produces data from the full dynamical system, however we only use the time series corresponding to the first variable. Documentation listing all the systems used as well as the default parameters can be found in Appendix B.1.

5.5.1 Within a System

Within a given dynamical system, we can attempt to differentiate between chaotic and periodic behavior. For a simple first experiment, we can look at one chaotic and one periodic time series and compare the results of our method. To start we will look at the two dynamical systems already introduced in earlier chapters, the Rössler system (defined in Eqn. 4.2) and the Lorenz system (defined in Eqn. 2.7). Figure 5.7 shows examples of a periodic and chaotic time series from each of these systems as well as the corresponding persistence diagrams. The three scores for each of these examples are listed in Table 5.1. Note that in the periodic case for the Lorenz system, there is one 1-dimensional persistence point so total persistence and maximum persistence are the same. It is clear that the persistence diagrams from the chaotic time series have more points than those from the periodic time series. This is reflected in the scores as the total persistence is higher in the chaotic case than in the periodic case for both example systems. Maximum persistence and persistent entropy have a same pattern where chaotic examples have higher scores than the periodic case. However the chaotic Lorenz example has a similar maximum persistence score to the periodic Rössler example.

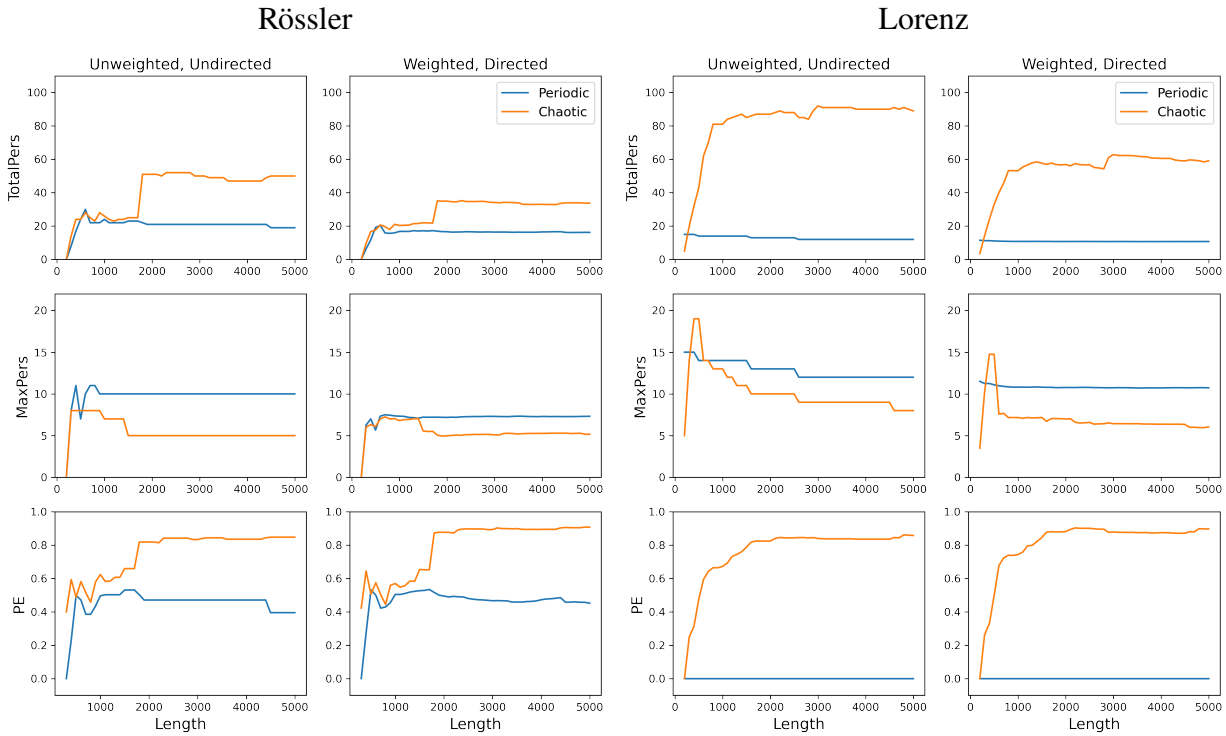


Figure 5.8: Total persistence, maximum persistence and persistent entropy vs. the length of the time series for the Rössler and Lorenz systems. First and third columns are the results using unweighted, undirected ordinal partition networks, while the second and fourth columns are using our weighted, directed version.

5.5.2 Impact of the Length of the Time Series

In the examples shown in Fig. 5.7, all the time series are the same length. Increasing the length of a periodic signal should not have a significant impact on the ordinal partition network or the resulting persistence diagram as the same pattern in the time series will keep repeating. However in a chaotic system where the pattern is not repeating, new permutations could appear causing new connections and additional loops in the ordinal partition network, thus changing the persistence diagrams and resulting statistics. We test these same systems computing all three statistics at a range of different lengths to see how long of a time series is necessary to detect differences as well as ensure that increasing the length will not cause the statistics to change too drastically. This is done by varying the parameter `SampleSize` in the `teaspoon` dynamical systems library. We vary this parameter between 200 and 5000 in increments of 100, at each step calculating all three scores. Here we also compare how the unweighted, undirected method performs. Figure 5.8 shows the plots of each

statistic in comparison to the length of the time series for the Rössler and Lorenz systems.

First looking at the Lorenz system, all three scores are fairly consistent for the periodic time series, but the total persistence and persistent entropy increase as the length of the chaotic time series increases. This is beneficial as it means increasing the length of the time series will only further differentiate the scores between the two classes. In comparison, maximum persistence does not appear to distinguish between the classes as clearly since at some lengths, the maximum persistence of the chaotic example is higher than that of the periodic example, and at other lengths the opposite is true. The Rössler system is slightly less clean. In the case of all three scores, the values are similar for both the periodic and chaotic time series up until around length 2000. This indicates a certain amount of time is needed before we can distinguish between the two behaviors. After a length of 2000, the scores are consistently different between the two classes.

5.5.3 Across Systems

Further, we test our methods ability to distinguish periodic and chaotic behavior across different systems. We generate one periodic and one chaotic time series from 28 different dynamical systems using `teaspoon`. The full list of systems and parameters for those systems can be found in Appendix B.1. For all systems, the `SampleSize` parameter is fixed at 2000 to ensure all time series are the same length. Figure 5.9 shows a histogram of each score for both the unweighted, undirected case, as well as the weighted, directed case. Ideally, we would like no overlap in the scores for the periodic time series and the chaotic time series. Total persistence separates the two classes the best, with only a few chaotic samples falling in bins with mostly periodic samples. However, the maximum persistence and persistent entropy scores are much more mixed. This indicates that total persistence is the best of the three scores for differentiating the classes. The exact scores from these experiments are listed in Tables B.3, B.4 and B.5 in Appendix B.2.

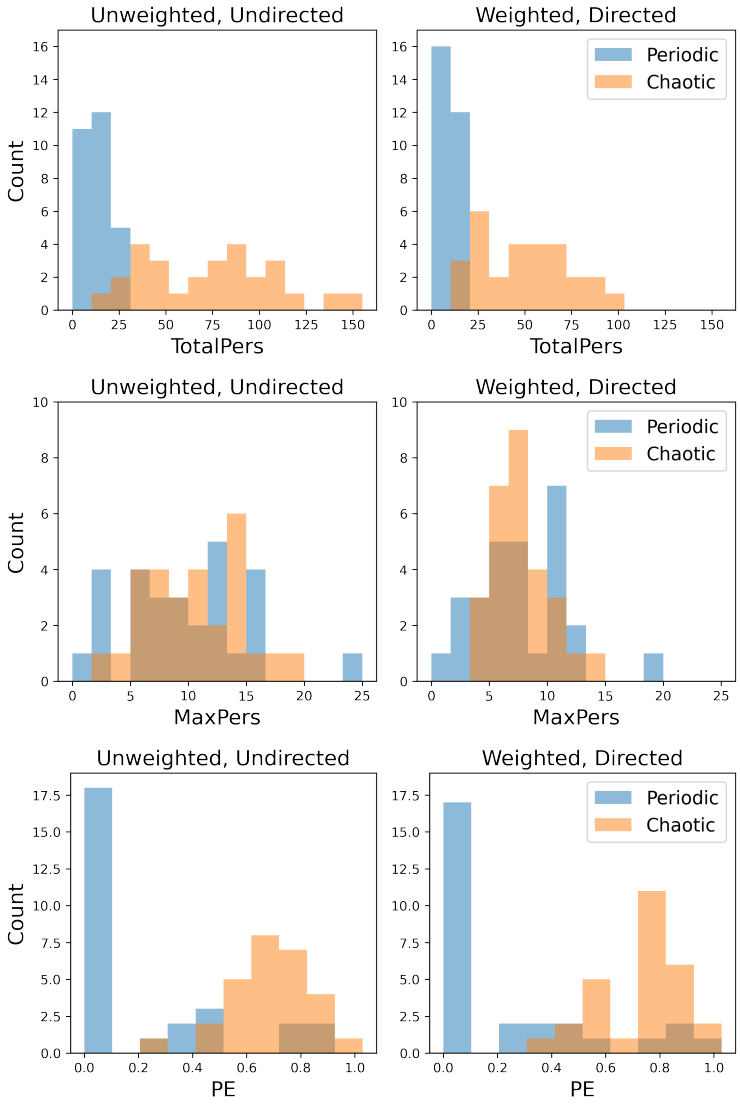


Figure 5.9: Histograms of total persistence, maximum persistence and persistent entropy for one periodic and one chaotic sample from 28 different dynamical system.

5.5.4 Size of the Graphs

As mentioned in Sec. 5.3, the number of vertices in the network is at most $d!$ for a given embedding dimension d, however not all permutations appear in practice. Figure 5.10 shows a histogram of how many vertices are in the networks from the experiment in Sec. 5.5.3. Note that while using dimension $d = 6$, there are 720 possible permutations, however in almost all cases less than 250 appear, and at most 264 appear. Thus the networks are significantly smaller than the worst case scenario, so choosing a higher dimension is still very computationally feasible.

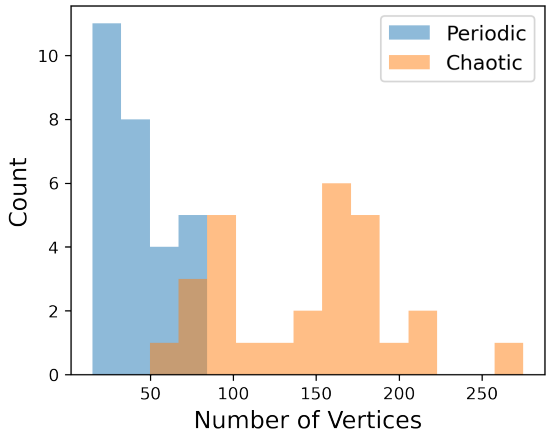


Figure 5.10: Histogram of the number of vertices in each network.

5.5.5 Robustness to Noise

The last experiment is to test how well the methods work in the presence of noise. We perform the same experiment as in Sec. 5.5.3 where we add noise to each time series at various levels.

Figure 5.11 shows the results at several signal-to-noise ratios (SNRs) for each of the three scores. Note that a higher SNR is a lower noise level, so in this plot noise increases going from left to right. Each point represents one sample, so all points above 50, for example, are from time series with noise added at an SNR of 50. The solid lines represent the mean score at each noise level, while the shaded region shows the standard deviation. Comparing each of the three scores, it is immediately clear that maximum persistence performs very poorly. There is no differentiation between the classes and the means are very similar. Looking at persistent entropy, as noise increases the means become more similar and the standard deviation intervals overlap more and more. This overlap indicates that with the addition of any amount of noise, PE is not a useful score.

The results of this experiment indicate total persistence is the most useful statistic. At SNR levels up until 25, the means between the periodic and chaotic samples are different and the standard deviations do not overlap. While there are some points visible that fall in the range of the opposite class, it gives a better separation between the two classes than either other score.

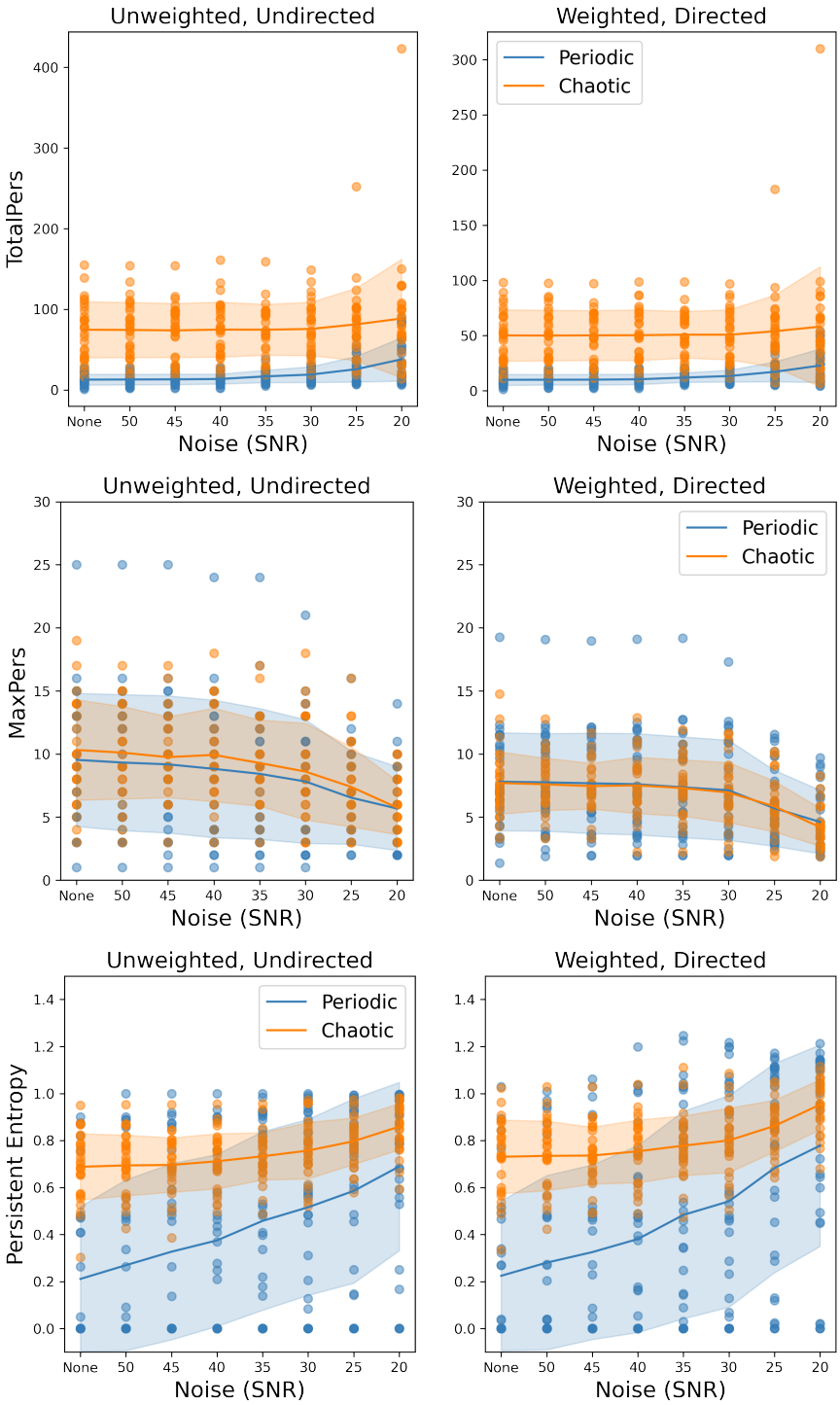


Figure 5.11: Plots of total persistence, maximum persistence and persistent entropy vs. noise level for one periodic and one chaotic sample from 28 different dynamical system. Each point is one sample, the solid line represents the mean and shaded region represents the standard deviation within a given class across the various noise levels. Note that the results at "None" are the same as those from Fig. 5.9.

5.6 Future Work

Here we provided an introductory exploration of the idea of incorporating the directed, weighted information into the topological analysis of ordinal partition networks. Thus far, it is unclear how much added value incorporating this information provides. A more thorough investigation of these ideas is needed, however this provides a jumping off point for future work. Here we outline some of our next steps as we will continue this work going forward.

Through this method there were several choices made regarding how to incorporate the transition counts and the directionality. The methods we chose worked well in our experiments, but in other applications different choices may be more effective. A more exhaustive analysis of weighting options would provide a more complete analysis of this method. For example, we set the weight of self loops to be 0.5, indicating a 50% chance of remaining in a given permutation. This choice worked the best in practice but was not theoretically grounded in any way. Additionally, as mentioned above, our weighting method is inspired by the graph diffusion distance however using a modification of this distance metric applied to directed graphs could provide a slightly different weighting scheme. Further, the interpretation that edges used more frequently are more important may not be entirely accurate, as noise could cause rapid changes back and forth between two permutations, thus artificially increasing the count.

There are also other methods of incorporating directed information into a simplicial complex. The Dowker complexes are well grounded theoretically as there is the duality in the source and sink filtrations as defined in Eqn. 5.7, but there are many additional options [104, 107]. One method developed specifically for applications in neuroscience is a directed flag complex [108, 109]. In this method, simplices are defined to be directed; a directed p-simplex is a $(p+1)$-tuple $(v_0, \ldots, v_p)$ such that for each $0 \leq i < j \leq p$ there is an edge from v_i to v_j. A p-simplex is now characterized by an ordered sequence of vertices, not just the collection of vertices it is built on. For example, the 2-simplices (v_0, v_1, v_2) and (v_1, v_0, v_2) are built on the same vertices but are distinct in the directed flag complex. Using this method, the simplicial complexes are larger, meaning they have more simplices, than those built using Dowker complexes since ordering of the vertices matters. There

is existing code for computing persistent homology based on the directed flag complex [110, 111]. A similar method is defined in [112] however it allows for repeated vertices (i.e. (v_0, v_0, v_1) is a 2-simplex), but this is computationally infeasible as you can have infinite repeats creating infinite possible simplices.

We tested three relatively simple scores to characterize persistence diagrams resulting from these methods. However these methods lose a lot of detail by aggregating all the information into one number. In the future, we will explore larger scale experiments such as the Rössler experiment in Sec. 4.3.3. Larger data sets will allow for the use of machine learning approaches such as template functions [1] and the many other persistence diagram featurization methods listed in Chapter 4. We also plan to directly compare the ordinal partition network approach to a time delay embedding approach to determine which performs better and which is more computationally feasible.

While periodic and chaotic behavior has been explored here as well as in other applications of ordinal partition networks, as far as we are aware quasi-periodicity has never been examined. It has been well studied in the context of using persistent homology and the time delay embedding [10], but future work could explore how this behavior appears in the networks.

CHAPTER 6

HOPF BIFURCATION DETECTION USING ZIGZAG PERSISTENCE

In existing methods, the standard process to analyze a collection of time series is by embedding each one into a point cloud as described in Sec. 2.3, followed by the computation of persistent homology. However, this requires an analysis of a collection of persistence diagrams. We develop a one-step method to analyze a collection of ordered point clouds using a modified version of persistence. The basic idea is to develop a filtration that utilizes the ordering of the point clouds. However, since we may not have a way of naturally nesting complexes built on the sequence of point clouds, we will use zigzag persistence. The work in this chapter was published in [29].

6.1 Zigzag Persistence

In the case of standard persistent homology as defined in Sec. 2.2, we assume that given our starting data, we can create a filtration that satisfies the inclusions as defined in Eqn. 2.1. These inclusions then induce maps on the homology groups, as described in Eqn. 2.2. Zigzag persistence considers the case where we have inclusions, but they do not all necessarily go in the same direction. These inclusions still induce linear maps on the homology groups, where the direction of the maps matches the direction of the inclusion. For example, given four simplicial complexes with inclusions as follows,

$$\mathcal{K}_1 \hookrightarrow \mathcal{K}_2 \hookleftarrow \mathcal{K}_3 \hookleftarrow \mathcal{K}_4$$

the induced maps on the homology groups are

$$H_k(\mathcal{K}_1) \rightarrow H_k(\mathcal{K}_2) \leftarrow H_k(\mathcal{K}_3) \leftarrow H_k(\mathcal{K}_4).$$

In this setting, standard persistent homology cannot be applied. This section will go over the basic background of zigzag persistence, following closely the presentation given in [113].

To start, a zigzag module, $\mathbb{V}$, of vector spaces is defined as a sequence of vector spaces and

linear maps

$$V_1 \overset{p_1}{\longleftrightarrow} V_2 \overset{p_2}{\longleftrightarrow} \cdots \overset{p_{n-1}}{\longleftrightarrow} V_n$$

where $\overset{p_i}{\longleftrightarrow}$ is either a forward map $\overset{f_i}{\longrightarrow}$ or backward map $\overset{g_i}{\longleftarrow}$. Specifically, the vector spaces are the homology groups in our application, and the linear maps are the induced maps.

A zigzag module's length is defined as the number of vector spaces n and the type τ is the sequence of f or g symbols indicating the linear maps in order from left to right. For example, the module

$$V_1 \overset{f_1}{\longrightarrow} V_2 \overset{g_2}{\longleftarrow} V_3 \overset{g_3}{\longleftarrow} V_4$$

has length 4 and is of type $\tau = fgg$. Note that persistent homology as described in Sec. 2.2 is just a special case where the type is $\tau = ff\cdots f$.

A submodule $\mathbb{W}$ of a τ-module $\mathbb{V}$ is defined by subspaces $W_i \leq V_i$ such that $f_i(W_i) \leq W_{i+1}$ or $g_i(W_{i+1}) \leq W_i$ for all i. Note that this condition guarantees that $\mathbb{W}$ is itself a τ-module with maps given by restrictions $f_i|_{W_i}$ or $g_i|_{W_{i+1}}$. A submodule $\mathbb{W}$ is called a summand of $\mathbb{V}$ if there exists a submodule $\mathbb{X} \leq \mathbb{V}$ where $V_i = W_i \oplus X_i$ for all i. Then we say $\mathbb{V} = \mathbb{W} \oplus \mathbb{X}$. We also need to be careful defining maps for the direct sum of two summands. Given a direct sum $\mathbb{V} \oplus \mathbb{W}$ with spaces $V_i \oplus W_i$, the maps are defined as $f_i \oplus h_i$ or $g_i \oplus k_i$, where f and g are forward and backward maps of $\mathbb{V}$ respectively, and h and k are forward and backward maps of $\mathbb{W}$ respectively. One type of module is called an interval module. Let τ be a type of length n, b, d be integers such that $1 \leq b \leq d \leq n$, and k be a field[1]. The τ-interval module with birth time b and death time d, denoted $\mathbb{I}_\tau(b, d)$ is defined with spaces

$$I_i = \begin{cases} \mathsf{k} & \text{if } b \leq i \leq d \\ 0 & \text{otherwise.} \end{cases}$$

with identity maps between adjacent copies of k and zero maps otherwise.

A τ-module is decomposable if it can be written as a direct sum of nonzero submodules in this way, and indecomposable otherwise. Any τ-module $\mathbb{V}$ has a Remak decomposition, i.e. $\mathbb{V} = \mathbb{W}_1 \oplus \mathbb{W}_2 \oplus \cdots \oplus \mathbb{W}_N$ where all $\mathbb{W}_j$ are indecomposable. Interval modules are in fact the only

[1] As with standard persistent homology, commonly k is chosen to be $\mathbb{Z}_2$.

indecomposable modules. These decompositions are not unique, but the Krull-Schmidt principle says the decompositions are unique up to reordering. Gabriel's theorem [114, 115], stated in Thm. 1, tells us that every τ-module can be written as a direct sum of interval modules.

Theorem 1 (Gabriel's Theorem) *The indecomposible τ-modules are precisely the intervals $\mathbb{I}(b, d)$ where $1 \leq b \leq d \leq n$ =length(τ). Equivalently, every τ-module can be written as a direct sum of intervals.*

From this information we can extract the zigzag persistence diagram.

For a zigzag module $\mathbb{V}$, the zigzag persistence diagram of $\mathbb{V}$ is the multiset

$$\text{Pers}(\mathbb{V}) = \{[b_j, d_j] \subset \{1, \ldots, n\} \mid j = 1, \ldots N\}$$

of integer intervals derived from a decomposition $\mathbb{V} \cong \mathbb{I}(b_1, d_1) \oplus \cdots \oplus \mathbb{I}(b_N, d_N)$. While this sets up a framework for any type τ, for the purposes of our work, we are going to limit ourselves to a particular type, $\tau = fgf \cdots gfg$, where we alternate forward and backward maps.

One benefit is zigzag persistence can be applied to capture changing shape through a collection of point clouds by using their unions as intermediate steps. Given an ordered collection of point clouds, $X_0, X_1, \ldots, X_n$, we can define a set of inclusions,

$$\begin{array}{ccccccccc} X_0 & & X_1 & & X_2 & \cdots & X_{n-1} & & X_n. \\ & \searrow \quad \swarrow & & \searrow \quad \swarrow & & & & \searrow \quad \swarrow & \\ & X_0 \cup X_1 & & X_1 \cup X_2 & & & & X_{n-1} \cup X_n & \end{array} \tag{6.1}$$

However, these are all still point clouds which have uninteresting homology since they are collections of disconnected points. In order to add additional structure at each step, we can compute the Rips complex (defined in Sec. 2.2.1) of each point cloud for a fixed threshold or radius, r. This results in the set of inclusions,

$$\begin{array}{ccccccccc} R(X_0, r) & & R(X_1, r) & & R(X_2, r) & \cdots & R(X_{n-1}, r) & & R(X_n, r). \\ & \searrow \quad \swarrow & & \searrow \quad \swarrow & & & & \searrow \quad \swarrow & \\ & R(X_0 \cup X_1, r) & & R(X_1 \cup X_2, r) & & & & R(X_{n-1} \cup X_n, r) & \end{array} \tag{6.2}$$

Computing the 1-dimensional homology of each complex in Eqn. 6.2 will result in a zigzag diagram of vector spaces and induced linear maps,

$$\begin{array}{ccccccccc} H_1(R(X_0,r)) & & H_1(R(X_1,r)) & \cdots & H_1(R(X_{n-1},r)) & & H_1(R(X_n,r)). \\ & \searrow & \swarrow & & & \searrow & \swarrow \\ & H_1(R(X_0 \cup X_1,r)) & & & & H_1(R(X_{n-1} \cup X_n,r)) & \end{array} \tag{6.3}$$

Computing zigzag persistence of this diagram will allow us to track loops that persist through the zigzag of simplicial complexes. We can generalize this idea to use a different radius for each Rips complex, $R(X_i, r_i)$. For the unions we choose the maximum radius between the two individual point clouds, $R(X_i \cup X_{i+1}, \max\{r_i, r_{i+1}\})$, to ensure the inclusions hold.

Zigzag persistence diagrams have a different interpretation than standard persistence diagrams. In the case of the Vietoris-Rips filtration, the birth and death times represent the scale at which features appear and disappear. However in zigzag persistence, they instead represent the portion of the zigzag where a homologous feature appears and disappears. For example, in the top example in Fig. 6.1, $\mathcal{K}_0$, $\mathcal{K}_0 \cup \mathcal{K}_1$ and $\mathcal{K}_1$ are all homologous. Thus, this zigzag of simplicial complexes has a 1-dimensional feature born in $\mathcal{K}_0$, which dies in $\mathcal{K}_1 \cup \mathcal{K}_2$. This corresponds to a 1-dimensional zigzag persistence point $(0, 1.5)$. However in the bottom example, the loops in $\mathcal{K}_0$ and $\mathcal{K}_1$ are separate and not homologous, thus there is a 1-dimensional feature born in $\mathcal{K}_0$ that dies in $\mathcal{K}_1$, and another that is born in $\mathcal{K}_0 \cup \mathcal{K}_1$ that dies in $\mathcal{K}_2$. This example would have two points in the 1-dimensional zigzag persistence diagram, $(0, 1)$ corresponding to the blue loop and $(0.5, 2)$ corresponding to the orange loop.

6.2 Bifurcations using ZigZag (BuZZ)

We can now present our method, Bifurcations using ZigZag (BuZZ), for combining the above tools to detect changes in dynamical systems, specifically changes that appear as a change in circular structure in the solution. We will focus on Hopf bifurcations [116], which are seen when a fixed point loses stability and a limit cycle is introduced. These types of bifurcations are particularly topological in nature, since varying one parameter in the system can cause the solution to the dynamical system to change from a small cluster, to a circular structure, and sometimes reduces

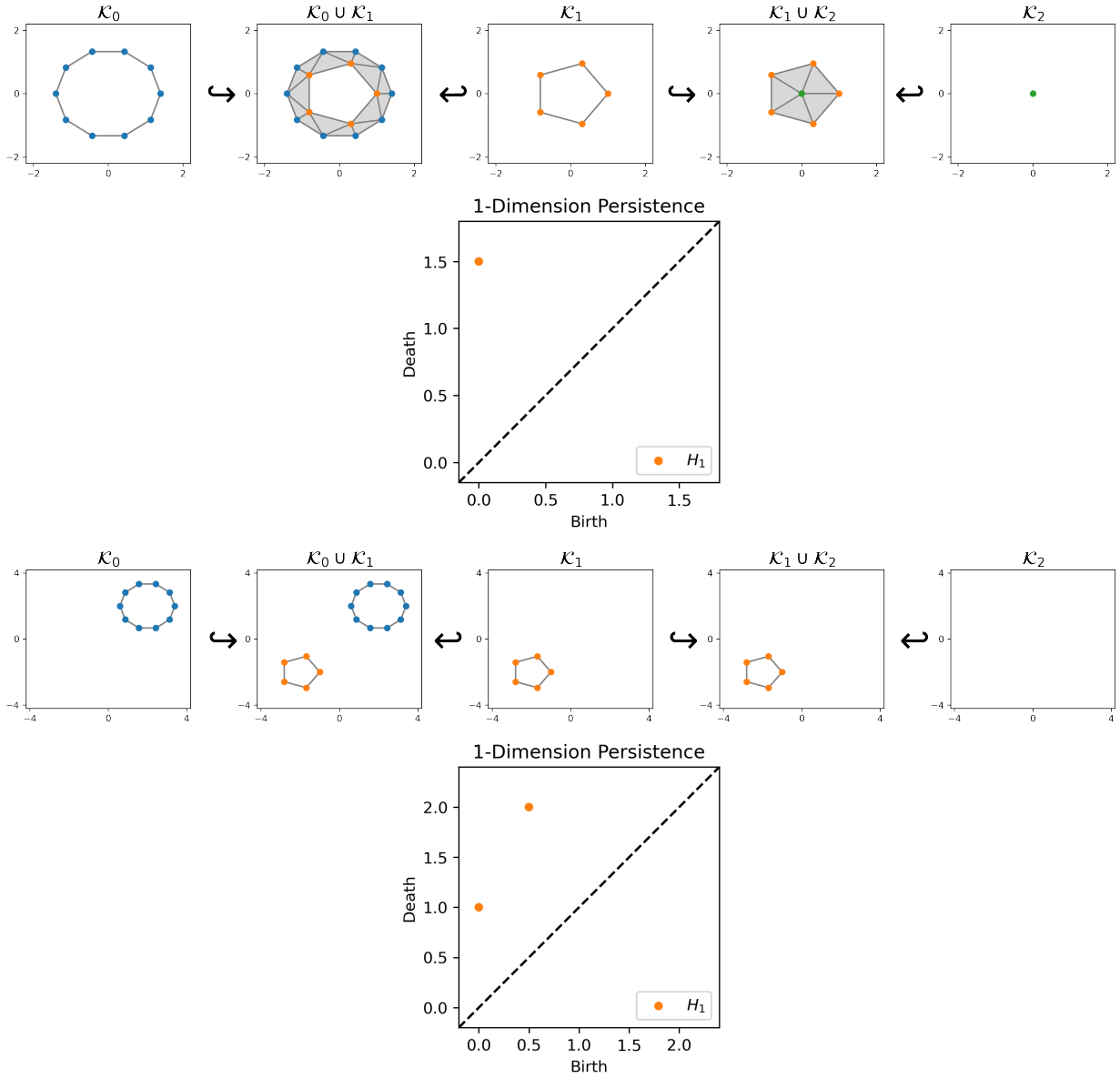


Figure 6.1: Examples of two zigzags of simplicial complexes with their corresponding 1-dimensional persistence diagrams.

back to a cluster. While persistent homology has been used to study Hopf bifurcations [117–120], none of the previous work uses zigzag persistence.

The necessary data for our method is a collection of time series for a varying input parameter value, as shown in Fig. 6.2(a). This particular example is a collection of time series given by $\{a\sin(t)\}$ for $a = 0.5, 1.0, 1.5, 2.0$ (going from top to bottom). Each time series is then embedded using the time delay embedding (shown in Fig. 6.2(b) using $d = 2$ and $\tau = 3$). While in general, the

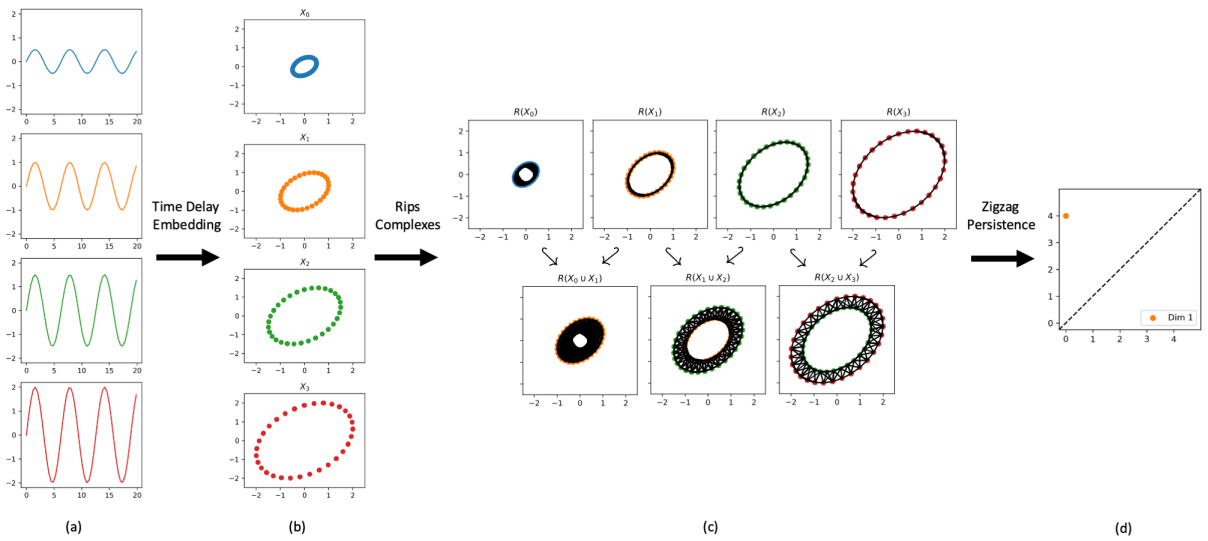


Figure 6.2: Outline of BuZZ method. The input time series is converted to an embedded point cloud via the time delay embedding. The Rips complexes are constructed for either a fixed r or a choice of r_i for each point cloud. Then, the zigzag persistence diagram is computed for the collection.

delay could be varied for each time series, the embedding dimension needs to be fixed so that each time series is embedded in the same space. For the sake of interpretability and visualization, we will use a dimension of $d = 2$ throughout this chapter. Sorting the resulting point clouds based on the input parameter value, the zigzag filtration can be formed from the collection of point clouds, as shown in Fig. 6.2(c). Lastly, computing zigzag persistence gives a zigzag persistence diagram, as shown in Fig. 6.2(d), encoding information about the structural changes moving through the complexes.

With the right choices of parameters, the 1-dimensional persistence point with the longest lifetime in the zigzag persistence diagram will have birth and death time corresponding to the indices in the zigzag where the Hopf bifurcation appears and disappears. Mapping the birth and death times back to the parameter values used to create the corresponding point clouds will give the range of parameter values where the Hopf bifurcation occurs.

Note that there are several parameter choices that need to be selected during the course of the BuZZ method. First, the dimension d and delay τ for converting each time series into a point cloud.

Fortunately, as mentioned in Sec. 2.3, there is a vast literature from the time series analysis for this, which leads to standard heuristics, some of which are published in [37–42]. The second and more difficult parameter to choose is the radius (or radii) for the Rips complexes. In this work, the given examples are simple enough that the choice of radii in the BuZZ method can be tuned by the user. In the tuning process, the user should select a radius that allows points in the circular structure to be connected, but not so large that it fills in the circle, or makes the Rips complexes too large to be computationally feasible.

6.3 Algorithms

While zigzag persistence has been in the literature for a decade, it has not often been used in application, and thus the software that computes it is not as efficient as software to compute standard persistent homology. A C++ package with python wrappers, `Dionysus 2`[2], has implemented zigzag persistence, however, it requires significant preprocessing to create the inputs. We have python code that, provided the collection of point clouds and radii, will perform all the necessary preprocessing to set up the zigzag diagram as shown in Eqn. 6.2 to pass as inputs to Dionysus. This code has been integrated into the python package `teaspoon`.

Dionysus requires two inputs, a list of simplices, `simplex_list`, and a list of lists, `times_list`, where the `times_list[i]` consists of a list of indices in the zigzag where the simplex, `simplex_list[i]`, is added and removed. A small example is shown in Fig. 6.3. Looking at that example, the two vertices and one edge in $R(X_0)$ appear at time 0, and disappear at time 1. There are two edges and a triangle in $R(X_0 \cup X_1)$ that appear there at time 0.5 (recalling that $R(X_i \cup X_{i+1})$ is time $i+0.5$) and disappear at time 1. Lastly, the one vertex in $R(X_1)$ appears at time 0.5, and never disappears in the zigzag sequence, so by default we set death time to be 2, which is the next index beyond the end of the zigzag sequence. This is done to avoid persistence points of the form (i, ∞), as our zigzag sequences are always finite and these points have no additional meaning. Note there are other special cases that can occur. If a simplex is added and removed

[2] https://www.mrzv.org/software/dionysus2/

Algorithm 6.1: Algorithm for preparing inputs for Dionysus to compute zigzag persistence using a fixed radius.

Input: X = $[X_0, X_1, \ldots, X_n]$ *# list of point clouds*
Input: r *# radius for Rips complexes*
A = GetRipsCplx(X[0], r) *# Handle special case of X_0 first*
simplex_list = A *# Initialize simplex list*
times_list = [[0,1] for _ in range(len(A))]
for $i = 1, \ldots, n$ **do**
    rips_complex = GetRipsCplx(X[i-1] ∪ X[i], r) *# Compute Rips complex of union*
    B = GetVertsList(X[i]) *# Initialize with vertices in X_i*
    M = [] *# Initialize for simplices who have vertices in X[i-1], X[i]*

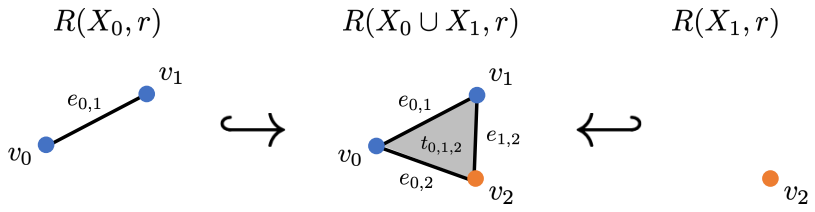
    for *simplex* ***in*** *rips_complex* **do**
        if *Get_0_Skeleton(simplex) ∩ A == Boundary(simplex)* **then**
            Continue *# handled in the initialization or previous iteration*
        else if *Get_0_Skeleton(simplex) ∩ B == Boundary(simplex)* **then**
            B.append(simplex) *# Simplices who only have vertices in X[i]*
        else
            M.append(simplex) *# Simplices who have vertices in both X[i-1] and X[i]*
    end
    # Update simplex_list, times_list for simplices who only have vertices in X[i]
    simplex_list = simplex_list + B
    times_list = [[i-0.5, i+1] for _ in range(len(B))]

    # Update simplex_list, times_list for simplices who have vertices in both X[i-1] and X[i]
    simplex_list = simplex_list + M
    times_list = [[i-0.5, i] for _ in range(len(M))]

    # Reinitialize for next iteration
    verts = verts_next
    A = B
end

multiple times, then the corresponding entry in `times_list` has more than two entries, where the even entries in the list correspond to when it appears, and the odd entries correspond to when it disappears. An example with this special case is shown in Fig. 6.4 and will be described in more detail later.

If we are using a fixed radius across the whole zigzag, these inputs can be computed rather easily. The algorithm is written more formally in Algorithm 6.1, however we will describe it intuitively here.



$\texttt{simplex\_list} = [\langle 0\rangle, \langle 1\rangle, \langle 0,1\rangle, \langle 2\rangle, \langle 1,2\rangle, \langle 0,2\rangle, \langle 0,1,2\rangle]$
$\texttt{times\_list} = [[0,1],[0,1],[0,1],[0.5,2],[0.5,1],[0.5,1],[0.5,1]]$

Figure 6.3: Example zigzag using fixed radius with computed inputs for Dionysus.

In this setting, we only need to compute the Rips complex of the unions, $R(X_i \cup X_{i+1}, r)$, which can be done using the Dionysus package, and the list of simplices can be created by combining lists of simplices for all i, removing duplicates. Next, we will outline how to construct the times list. Starting with the set of simplices in $R(X_i \cup X_{i+1}, r)$, we can split them into three groups: (a) simplices for which all vertices are in X_i, (b) simplices for which all vertices are in X_{i+1}, or (c) simplices for which some vertices are in X_i and some are in X_{i+1}. Because of the construction of the zigzag, all simplices in group (a) appear at time $i - 0.5$ (since $R(X_i, r)$ also includes backwards into $R(X_{i-1} \cup X_i, r)$) and disappear at time $i + 1$ (since the union $R(X_i \cup X_{i+1}, r)$ is the last time simplices in X_i are included). Similarly, all simplices in group (b) appear at time $i+0.5$ (since this is the first time simplices X_{i+1} are included) and disappear at time $i+2$ (since $R(X_{i+1}, r)$ also includes forward into $R(X_{i+1} \cup X_{i+2}, r)$). Lastly, all simplices in group (c) exist only at $R(X_i \cup X_{i+1}, r)$, so they appear at time $i + 0.5$ and disappear at $i + 1$. Note that the first union of simplicial complexes, $R(X_0 \cup X_1, r)$, needs to be treated separately, since all vertices in group (a) will appear at 0, not $i - 0.5$ as is the case for $i > 0$.

One important note when looking at Algorithm 6.1 is that all vertices have to have a unique label. Thus the vertices in $R(X_0)$ are numbered $0, \ldots, |X_0| - 1$, where $|X_0|$ denotes the number of points in X_0. Next, the vertices in $R(X_1)$ are numbered $|X_0|, \ldots, |X_0|+|X_1|$, and so on. Since higher dimensional simplices are represented as lists of the vertices the simplex is built on, ensuring a consistent labeling is essential. Hence, in Lines 10 and 13 of the algorithm, the labeling of vertices (and thus the labeling of higher dimensional simplices) must be adjusted.

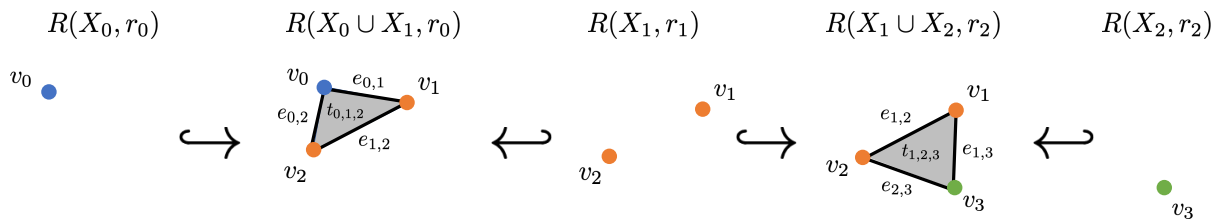


Figure 6.4: Example zigzag using a changing radius with computed inputs for Dionysus. In this example $r_0 > r_1$ and $r_2 > r_1$.

Using a varied radius, as described in Sec. 6.1, complicates the above procedure. Using the same radius, we are guaranteed all simplices in group (a) are in both $R(X_i, r)$ and $R(X_i \cup X_{i+1}, r)$, and similarly for group (b), thus we only need to compute $R(X_i \cup X_{i+1}, r)$. However, with a changing radius this is no longer true. In the example shown in Fig. 6.4, the edge $e_{1,2}$ appears in both $R(X_0 \cup X_1, r_0)$ and $R(X_1 \cup X_2, r_2)$ since $r_2 > r_1$ and $r_0 > r_1$, but it is not in $R(X_1, r_1)$. Thus, its corresponding list in `times_list` is $[0.5, 1, 1.5, 2]$. The inputs to Dionysus can be computed using the same method as above, except the Rips complex needs to be computed for each point cloud, not just the unions, and additional checks need to be done to make sure a simplex being added did not already appear and disappear once before. If it did, the entry in `times_list` needs to be extended to account for the newest appearance and disappearance.

Because of the additional Rips complex computations, and the checks for the special case, the case of a changing radius is significantly more computationally expensive than the case of a fixed radius. In both cases, there is the computational cost of the zigzag persistence computation as well. The computational complexity of zigzag persistence is $O(nm^2)$ where n is the number of simplices in the entire filtration and m is the number of simplices in the largest single complex [121].

To alleviate this, a radius for the Rips complexes should be selected so that it is as small as possible without breaking the circular structure and thus breaking the topology. Additionally, it needs to be chosen so that in the union of point clouds, no small circles are created in the region between the ring like structures, which would cause the larger circular structures to no longer be homologically equivalent. We acknowledge that choosing one radius value is fairly counter intuitive

since the standard Vietoris-Rips filtration is designed to avoid exactly this choice. In one of our experiments we test how sensitive the method is to the choice of r and in Sec. 6.5 we address a possible modification to the method to avoid choosing the radius r all together.

6.4 Results

We will test the BuZZ method on three different examples. The first example is not based on time series data, but is instead a simple proof-of-concept example to test our method's ability to detect changing circular behavior. The second example is based on synthetic time series data generated from noisy sine waves of varying amplitude. This lets us fully utilize the BuZZ method, including the time delay embedding, as well as test resiliency to noise. The last example is detecting a Hopf bifurcation in the Sel'kov model of glycolysis [122]. This is a well studied dynamical system and the range of parameter values for which a Hopf bifurcation occurs is known.

6.4.1 Synthetic Point Cloud Example

To start, we will consider a small, synthetic example generating point cloud circles of varying size as shown in Fig. 6.5. Note, because we are starting with point clouds, we skip the time delay embedding step for this example. While each point cloud is sampled from a circle, the first and last point clouds consist of relatively small circles. So the strongest circular structure we can see visually starts with X_1 and ends with X_3. This is the range we would like to detect using zigzag persistence.

For this example, we will use the generalized version of the zigzag filtration in Eqn. 6.2 using a changing radii. Computing zigzag persistence gives the persistence diagram shown in Fig. 6.5. Recall that birth and death times are assigned based on the location in the zigzag that a feature appears and disappears. Thus, the one-dimensional point $(1, 3.5)$ in the persistence diagram corresponds to a feature that first appears at $R(X_1)$ and last appears in $R(X_3)$. Thus, using the zigzag persistence diagram we can detect the appearance and disappearance of the circular feature.

This is clearly an overly simplified situation as each point cloud is sampled from a perfect circle.

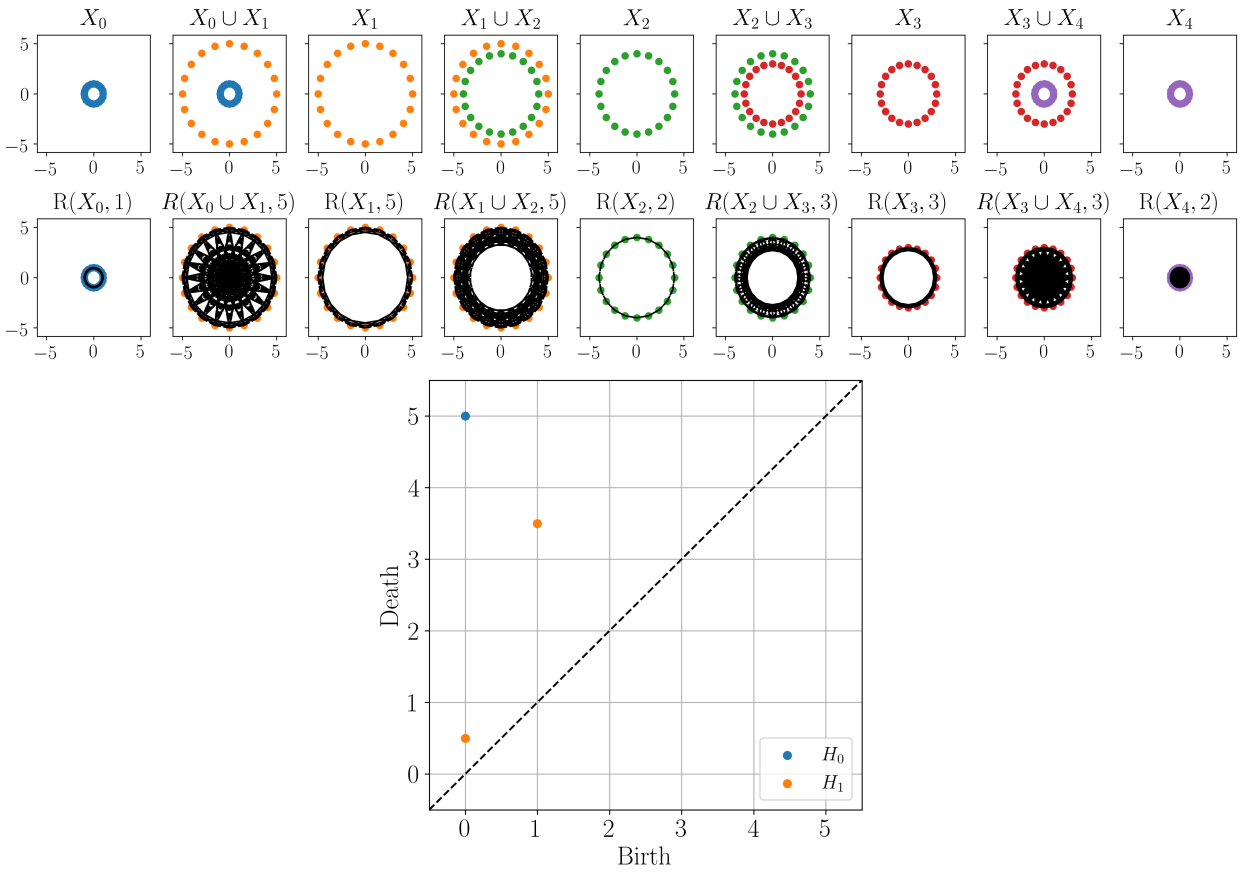


Figure 6.5: Top: Example zigzag of point clouds with unions considered in Sec. 6.4.2. Middle: Zigzag filtration applied to point clouds using the Rips complex with specified radii. Note that 2-simplices are not shown in the complexes. Bottom: The resulting zigzag persistence diagram.

Next, we will look at a more realistic example.

6.4.2 Synthetic Time Series Example

For the second example, we generate synthetic time series data and apply the full method described in Sec. 6.2. We start by generating sine waves of varying amplitudes and add noise drawn from uniformly from $[-0.1, 0.1]$. The time series are then each embedded using the time delay embedding with dimension $d = 2$, for the purpose of easy visualization, and delay $\tau = 4$. The time series and corresponding time delay embeddings are shown in Fig. 6.6. Looking at the time series, in the first and last time series any signal is mostly obscured by noise, resulting in a small clustered time delay embedding. However, for the other time series, the time delay embedding is still circular, picking

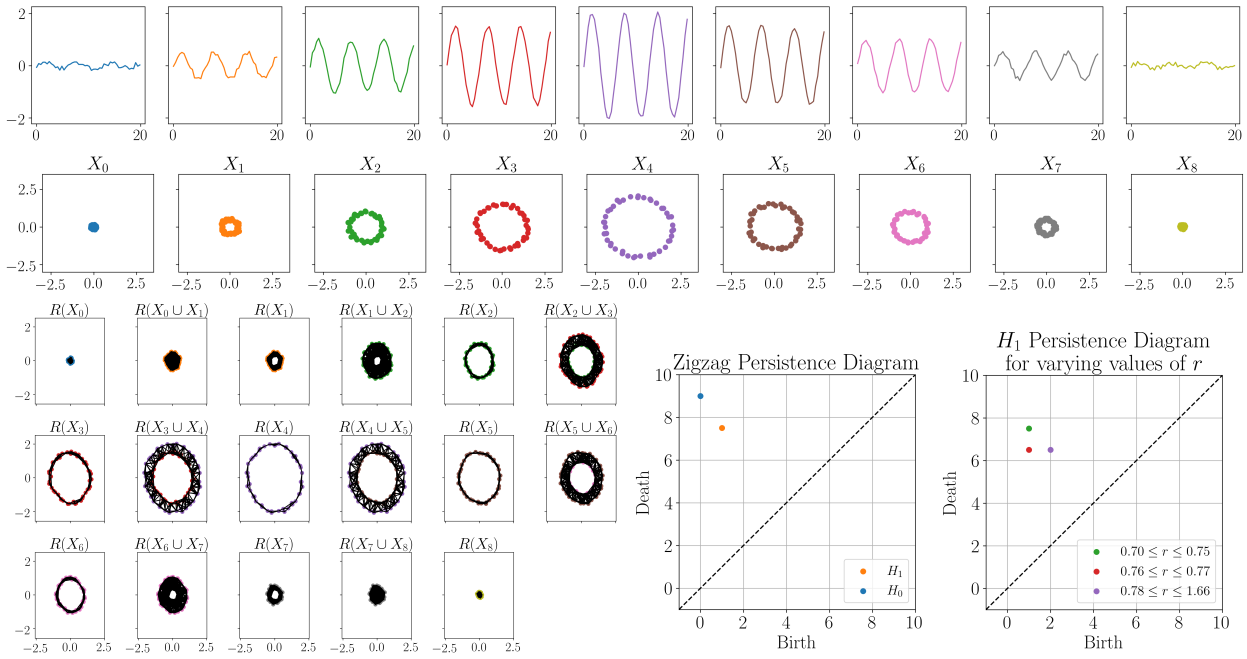


Figure 6.6: First and second rows: Generated time series data and corresponding time delay embeddings. Bottom left: The zigzag filtration using Rips complex with fixed radius of 0.72. Note that 2-simplices are not shown in the complexes. Bottom middle: The corresponding zigzag persistence diagram. Bottom right: Persistence diagram showing how the 1-dimensional off diagonal point varies depending on the Rips complex radius parameter choice.

up the periodic behavior even with the noise.

Next we compute zigzag persistence, resulting in the zigzag of Rips complexes and zigzag persistence diagram shown in Fig. 6.6. The zigzag persistence diagram has a one-dimensional point with coordinates $(1, 7.5)$, indicating the circular feature appears in $R(X_1)$, and disappears going into $R(X_8)$. This is the region that we visually see a circular feature, so the results match our intuition.

In this experiment, we also test the variation in the zigzag persistence diagram based on the choice of radius parameter. For this particular example, choosing too small of a radius ($r < 0.70$) introduces noisy circular features in between the circular structures in the union, causing the circles to no longer be homologically equivalent, and thus we do not get the desired one-dimensional persistence point. Choosing a radius $0.70 \leq r \leq 0.75$ resulted in the same zigzag persistence diagram shown in Fig. 6.6. However, increasing the radius into the range $0.76 \leq r \leq 0.77$, the one dimensional persistence point shifts to coordinates $(1, 6.5)$, thus we are not detecting the small

circular structure in X_7. Increasing the radius further into the range $0.78 \leq r \leq 1.66$, the one dimensional persistence point shifts to coordinates $(2, 6.5)$, thus we are not detecting the small circular structure in X_1 or X_7. This is visualized in the bottom right plot in Fig. 6.6. Continuing to increase the radius $r > 1.66$, will cause more of the circular structures to fill in, detecting a smaller and smaller range where the circular feature is detected. Thus, without careful selection of the radius parameter, smaller circular features may not be detected, however detection of the larger circular features is much more robust to parameter choices.

6.4.3 Sel'kov Model

Our last experiment aims to detect a bifurcation in the Sel'kov model [122], a model for glycolysis which is a process of breaking down sugar for energy. This model is defined by the system of differential equations,

$$\begin{aligned} \frac{dx}{dt} &= -x + ay + x^2 y, \\ \frac{dy}{dt} &= b - ay - x^2 y. \end{aligned}$$

In this system, x and y represent the concentration of ADP (adenosine diphosphate) and F6P (fructose-6-phosphate), respectively. This system has a Hopf bifurcation for select choices of parameters a and b. This limit cycle behavior corresponds to the oscillatory rise and fall of the chemical compounds through the glycolysis process.

For our experiments, we will fix $a = 0.1$ and vary the parameter b. We generate data using `odeint` in python, with time values $\{0, \ldots, 499\}$ and initial conditions $(0, 0)$. We also remove the first 50 points to remove transients at the beginning of the model (this is sometimes referred to as a "burn-in period"). The resulting data is shown in Fig. 6.7. This data is constructed using full knowledge of the model, however, in practice, typically only one measurement function is obtained through an experiment, and then the time-delay embedding is used to reconstruct the underlying system. To mimic this setup, we will only use the time series corresponding to the x-coordinates from the model and use the delay embedding. The time series corresponding to the x-coordinates

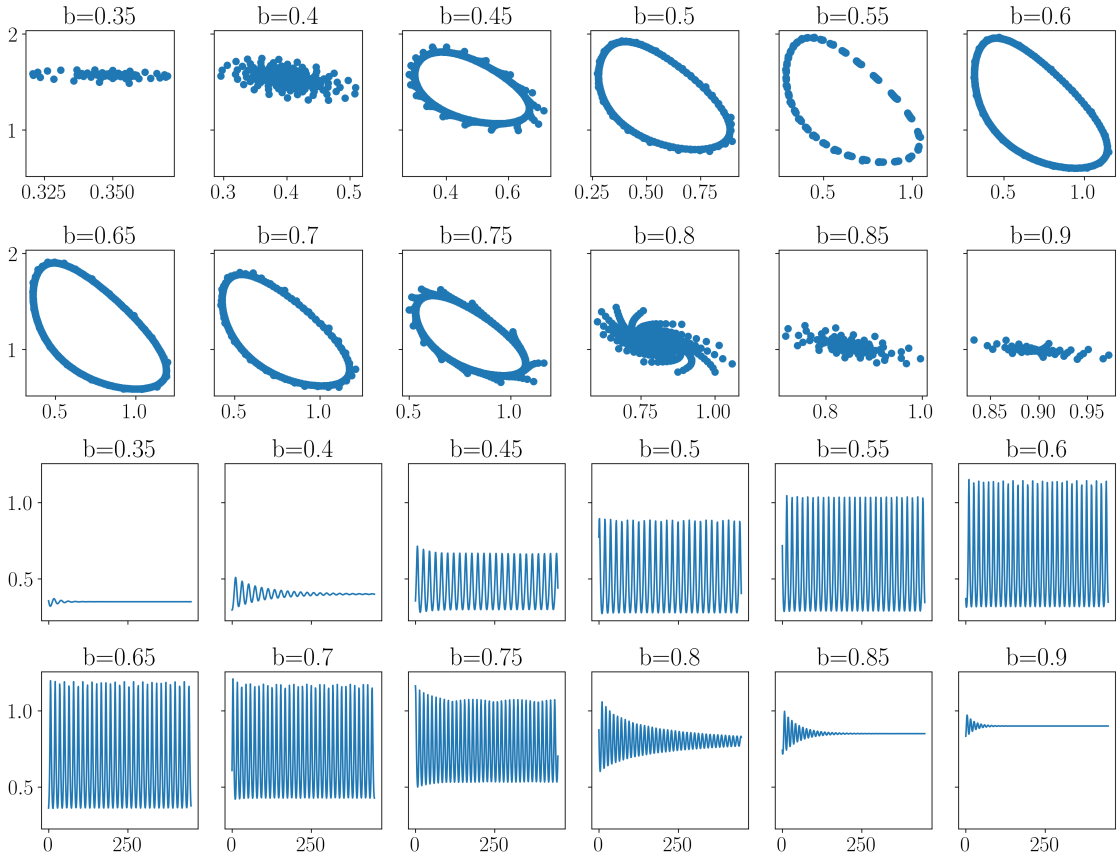


Figure 6.7: Top: Examples of samplings of the state space of the Sel'kov model for varying parameter value b. Bottom: Time series corresponding to only retaining the x-coordinates of the solutions shown in top figure.

are shown in Fig. 6.7. These time series are then embedded using the time delay embedding with dimension $d = 2$, for the purpose of easy visualization, and delay $\tau = 3$.

The next step would be to compute zigzag persistence as described in Sec. 6.1, however due to the large number of points in the time delay embeddings, this becomes computationally expensive. In order to reduce the computation time, we subsample these point clouds using the furthest point sampling method (also called a greedy permutation) [123]. This method works by iteratively selecting points to be included in the subsampled point cloud. It begins by selecting one starting point in the point cloud (by default we select the first point), then the next point added is the furthest point away from the starting point. The third point added is the furthest point away from both the first and second points, and so on. This process is iterated until the desired number of points is reached.

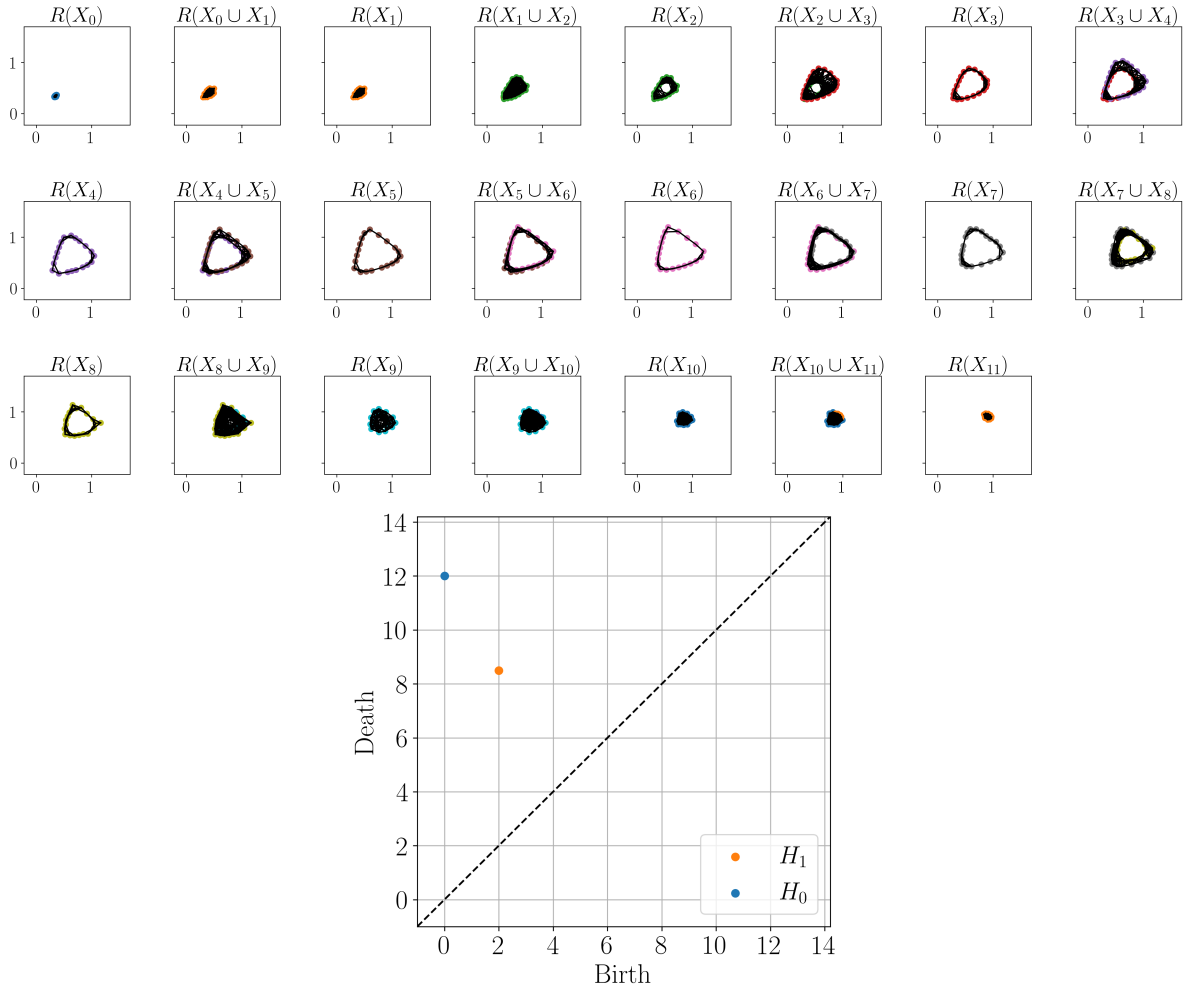


Figure 6.8: Top: zigzag filtration using Rips complex of the reconstruction with fixed radius of 0.25. Note that 2-simplices are not shown in the complexes. Bottom: resulting zigzag persistence diagram.

We subsample down to only 20 points in each point cloud, compute the Rips complex zigzag for a fixed radius value of 0.25, and then compute the zigzag persistence. Figure 6.8 shows the zigzag filtration of Rips complexes along with the resulting zigzag persistence diagram. In the zigzag persistence diagram, the point with the longest lifetime has coordinates $(2, 8.5)$. Again, since these coordinates correspond to the index in the zigzag sequence, this point corresponds to a feature appearing at $R(X_2)$ and disappearing at $R(X_8 \cup X_9)$. Looking back at which values of b were used to generate these point clouds, we see this corresponds to a feature appearing at $b = 0.45$ and disappearing at $b = 0.8$. For the fixed parameter value of $a = 0.1$, the Sel'kov model has a limit cycle approximately between the parameter values $0.4 \leq b \leq 0.8$ [88]. Our method is picking

up approximately that same range. These results use the x-coordinates of the model, however the same results can be obtained using the y-coordinates and a slightly larger radius value.

6.5 Future Work

The results for the BuZZ method are promising, and there are several future directions for this project. First we will present a more idealistic idea that currently cannot be implemented due to computational challenges, followed by a second approach that incorporates some of the methods from Chapter 5.

6.5.1 Multiparameter Zigzag Persistence

In the BuZZ method, when computing the Rips complexes, one has to make a choice of the radius parameter (or parameters). For the experiments shown, this parameter was hand selected. While in practice it was not difficult to choose a radius that worked, choosing one parameter value feels counter-intuitive in the context of persistent homology where in the Vietoris-Rips filtration where a range of parameter values is used. One method of incorporating this parameter into the framework is to use multiparameter persistence [124]. Standard persistence as defined in Sec. 2.2 captures changing topological structure over one varying parameter. If you have two changing parameters, you can define a bi-filtration,

$$\begin{array}{ccccccc}
\mathcal{K}_{a_0,b_m} & \hookrightarrow & \mathcal{K}_{a_1,b_m} & \hookrightarrow & \mathcal{K}_{a_2,b_m} & \hookrightarrow \cdots \hookrightarrow & \mathcal{K}_{a_n,b_m} \\
\uparrow & & \uparrow & & \uparrow & & \uparrow \\
\vdots & & \vdots & & \vdots & \iddots & \vdots \\
\uparrow & & \uparrow & & \uparrow & & \uparrow \\
\mathcal{K}_{a_0,b_1} & \hookrightarrow & \mathcal{K}_{a_1,b_1} & \hookrightarrow & \mathcal{K}_{a_2,b_1} & \hookrightarrow \cdots \hookrightarrow & \mathcal{K}_{a_n,b_1} \\
\uparrow & & \uparrow & & \uparrow & & \\
\mathcal{K}_{a_0,b_0} & \hookrightarrow & \mathcal{K}_{a_1,b_0} & \hookrightarrow & \mathcal{K}_{a_2,b_0} & \hookrightarrow \cdots \hookrightarrow & \mathcal{K}_{a_n,b_0}
\end{array} \tag{6.4}$$

where $a_0 \leq a_1 \leq a_2 \leq \cdots \leq a_n$ and $b_0 \leq b_1 \leq b_2 \leq \cdots \leq b_m$.

One downside of multiparameter persistence is that there are not well behaved summaries (like the persistence diagram for single parameter persistence), making it more challenging to use in

application; however, there are statistics on multiparameter persistence modules that can be useful in practice [124].

To incorporate this into our framework, we could use multiparameter zigzag persistence as in Eqn. 6.5 where $r_0 \leq r_1 \leq r_2 \leq \cdots$. In this example, each row is a zigzag complex as in Eqn. 6.2, while each column is a standard Vietoris-Rips filtration as in Eqn. 2.3.

$$\begin{array}{ccccccc}
\vdots & & \vdots & & \vdots & & \vdots \\
\uparrow & & \uparrow & & \uparrow & & \uparrow \\
R(X_0, r_2) & \hookrightarrow & R(X_0 \cup X_1, r_2) & \hookleftarrow & R(X_1, r_2) & \hookrightarrow & R(X_1 \cup X_2, r_2) \hookleftarrow \cdots \\
\uparrow & & \uparrow & & \uparrow & & \uparrow \\
R(X_0, r_1) & \hookrightarrow & R(X_0 \cup X_1, r_1) & \hookleftarrow & R(X_1, r_1) & \hookrightarrow & R(X_1 \cup X_2, r_1) \hookleftarrow \cdots \\
\uparrow & & \uparrow & & \uparrow & & \uparrow \\
R(X_0, r_0) & \hookrightarrow & R(X_0 \cup X_1, r_0) & \hookleftarrow & R(X_1, r_0) & \hookrightarrow & R(X_1 \cup X_2, r_0) \hookleftarrow \cdots
\end{array} \tag{6.5}$$

While there is software available to compute multiparameter persistent homology [125], it currently cannot compute multiparameter zigzag persistence to our knowledge. Thus, this type of framework is not feasible computationally given the current software available, it would provide useful insight if it becomes feasible in the future.

6.5.2 BuZZ-Net

As mentioned, the most significant bottleneck in the BuZZ pipeline is computation time. Longer time series lead to larger point clouds, which drives up the computational cost of computing Rips complexes and zigzag persistence. To combat this issue, we propose replacing the time delay embedding step with the ordinal partition networks. Thus, we would create a zigzag of networks, each of which can be turned into a simplicial complex using the clique complex. The number of nodes in the ordinal partition networks is fixed based on the chosen embedding dimension d, with up to $d!$ possible permutations, however we showed in Sec. 5.5.4 that when using $d = 6$, not even half the possible permutations are actually used in each network. Replacing Rips complexes with clique complexes of the ordinal partition networks would solve the issue with longer time series.

Further, we can make a slight modification to the method to speed up computation time. In Sec. 6.1, we introduce the zigzag of Rips complexes in Eqn. 6.2 using the union of the point clouds as intermediate steps. However, using a different choice we can leverage some additional theory of zigzag persistence.

First we must introduce some theoretical concepts needed. We will present these as in [113, Sec. 5.3]. For simplicity, starting with a zigzag of simplicial complexes, we can consider the following sequence,

$$\begin{array}{ccccccccccccc} & & & & & & A \cup B & & & & & & \\ & & & & & \nearrow & & \nwarrow & & & & & \\ X_0 & \longleftrightarrow & \cdots & \longleftrightarrow & X_{k-2} & \longleftrightarrow & A & & B & \longleftrightarrow & X_{k+2} & \longleftrightarrow & \cdots & \longleftrightarrow & X_n \\ & & & & & \nwarrow & & \nearrow & & & & & \\ & & & & & & A \cap B & & & & & & \end{array} \tag{6.6}$$

where $\longleftrightarrow$ are arbitrary maps. Let $\mathbb{X}^+$ be the upper zigzag diagram going up through $A \cup B$ and $\mathbb{X}^-$ be the lower zigzag diagram going down through $A \cap B$. Then the following theorem gives a complete bijection between points in the persistence diagram.

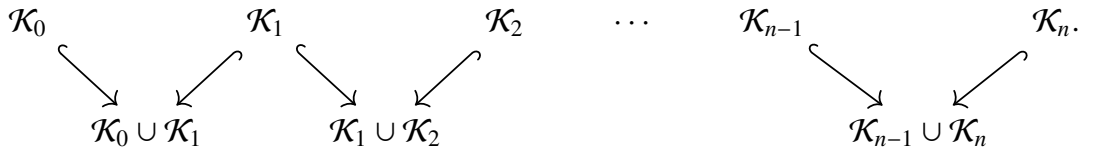
Theorem 2 (The Strong Diamond Principle) *Given $\mathbb{X}^+$ and $\mathbb{X}^-$ as above, there is a complete bijection between the points in the persistence diagrams $\{D_k(\mathbb{X}^+) \mid k = 0, 1, \ldots\}$ and the points in the persistence diagrams $\{D_k(\mathbb{X}^-) \mid k = 0, 1, \ldots\}$. The matching is defined by the following rules. First there are matchings across different dimension diagrams:*

- *$(k, k) \in D_{\ell+1}(\mathbb{X}^+)$ is matched with $(k, k) \in D_\ell(\mathbb{X}^-)$.*

The rest of the matchings remain within a given homological dimension:

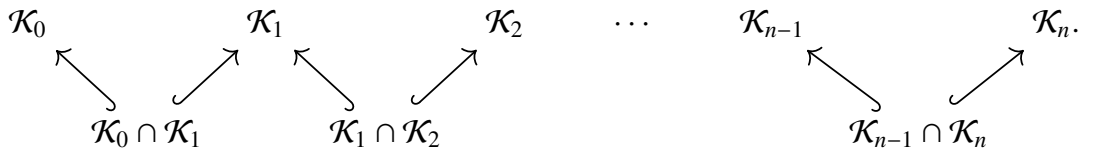
- *Type (b, k) is matched with type $(b, k-1)$ and vice versa, for $b \leq k-1$.*
- *Type (k, d) is matched with type $(k+1, d)$ and vice versa, for $d \geq k+1$.*
- *Type (b, d) is matched with type (b, d) in all other cases.*

Theorem 2 can then be used iteratively to map between points in the zigzag persistence diagrams of two different zigzag complexes, one using the unions and one using the intersections. If we have a sequence of simplicial complexes, $(\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_n)$, that are all built from the same vertex set then we can define the union zigzag as



(6.7)

Similarly, we can define the intersection zigzag as



(6.8)

We can index the zigzag complexes by $\{0, \frac{1}{2}, 1, 1\frac{1}{2}, \ldots, n\}$, which is the same approach taken in Sec. 6.2. Given persistence diagrams, $D_k^{\cap}$ and $D_k^{\cup}$, we can map points between them via the following rules:

- If $b, d \in \mathbb{Z}$, then $(b, d) \in D_k^{\cap}$ maps to $(b, d) \in D_k^{\cup}$.
- Points of type $\left(c\frac{1}{2}, c\frac{1}{2}\right) \in D_k^{\cap}$ maps to $\left(c\frac{1}{2}, c\frac{1}{2}\right) \in D_{k+1}^{\cup}$.
- Otherwise, $(b, d) \longleftrightarrow (b'd')$ where $\{b, b'\}$ is an unordered pair of the type $\left\{c\frac{1}{2}, c+1\right\}$ and $\{d, d'\}$ is an unordered pair of the type $\left\{c, c\frac{1}{2}\right\}$ and homological dimension remains fixed.

The ordinal partition networks as defined in Sec. 5.3 have vertices corresponding to permutations of the set $\{1, \ldots, d\}$ for a chosen dimension d. Thus a collection of ordinal partition networks with the same chosen dimension all have the same vertex set[3]. As described in Sec. 5.2, we can build a simplicial complex from an ordinal partition network using clique complexes. Returning to the beginning of the BuZZ framework where we have a sequence of time series, we can compute the ordinal partition network of each, then transform each into a simplicial complex. This results

[3]While not every time series uses the same set of permutations, each network is built from the same collection of possible permutations.

in a sequence of simplicial complexes built on a common vertex set, from which we can compute zigzag persistence of the intersection zigzag. This will be computationally easier and can be used to determine the zigzag persistence of the union zigzag, if needed. This would improve the BuZZ method by removing choices such as the radius (or radii) of the Rips complexes and it would speed up computation time. We call this combination of methods Bifurcations using ZigZags and Networks, or BuZZ-Net.

Further, we can consider how to incorporate the modifications to the ordinal partition networks presented in Chapter 5. It is not immediately obvious how to incorporate directed and weighted information in the zigzag persistence framework, however it may provide useful additional information.

CHAPTER 7

CONCLUSION

In this dissertation, we present four projects, each of which address an application of or method for topological time series analysis. We have shown that persistent homology is a useful tool in detecting a daily cycle in hurricanes, and maximum persistence, a simple statistic on persistence diagrams, is all that is needed to quantify it. Our approach has shown success across two different hurricanes and is robust to variations in temporal and spatial resolution. Further, it is fairly robust to the type of noise that is usually detrimental in persistent homology based approaches.

More complex approaches of quantifying structure in persistence diagrams can be done thorough machine learning methods. We modify the template function featurization method to use local regions in the persistence diagrams, reducing the overall number of features needed to capture the structure in the diagrams. The modified approach improves results on basic experiments as well as classifying periodic and chaotic behavior in time series data.

While the time delay embedding is a well studied and useful tool, the ordinal partition networks provide a complimentary summary of the underlying structure in a more compact data structure. They are also rich with detail as weighted, directed networks, so we explore incorporating additional features in the topological analysis. We conduct a range of preliminary experiments showing robustness of our approach both within a given dynamical system as well as across 28 different systems. Further we show that the methods are robust to the length of the time series and the addition of noise.

Lastly, we develop a data-driven approach to detecting Hopf bifurcations using zigzag persistence. While persistent homology has shown success in numerous application areas, thus far zigzag persistence is fairly underused. However, we show that it is well suited in the application to Hopf bifurcation detection and provide several examples where it works in practice.

There is significant work that can build off of the methods presented in this dissertation. We provide several directions for future work in the last two chapters to modify and improve upon the

methods developed here.

We also aim to make our methods accessible to the research community. The approaches developed in the last three chapters (the modified approaches to the template function featurization and the ordinal partition network, as well as the BuZZ method) are all implemented in the `teaspoon` open source python package [85]. This package contains numerous methods for topological signal processing, making our results more easily reproducible.

APPENDICES

APPENDIX A

TABLES OF RESULTS FROM ADAPTIVE TEMPLATE SYSTEMS

This appendix has the tables with the accuracies corresponding to Figure 4.8. Tables A.1 and A.2 have the results using tent functions and interpolating polynomial functions, respectively. The scores highlighted in green give the best average score across all testing columns; the scores highlighted in blue have overlapping intervals of standard deviation with the best score. A more thorough explanation and interpretation of these results is presented in 4.3.2.

Freq	MSK	Dgm0		Dgm1		Dgm0 & Dgm1 (Combined Partitioning)		Dgm0 & Dgm1 (Split Partitioning)	
		Train	Test	Train	Test	Train	Test	Train	Test
1	94.7 ± 5.1	97.1 ± 1.8	81.2 ± 4.2	93.9 ± 2.9	70.8 ± 4.4	99.4 ± 0.4	84.7 ± 2.2	100.0 ± 0.0	92.5 ± 2.0
2	99.3 ± 0.9	91.2 ± 0.9	74.8 ± 4.3	97.5 ± 0.8	73.2 ± 4.3	97.7 ± 0.7	79.0 ± 3.6	100.0 ± 0.0	91.8 ± 1.8
3	96.3 ± 2.2	80.4 ± 1.7	57.3 ± 6.7	94.9 ± 3.4	71.3 ± 4.0	97.4 ± 0.8	81.6 ± 2.4	98.8 ± 0.5	86.9 ± 3.3
4	97.3 ± 1.9	62.9 ± 2.5	39.5 ± 5.5	94.8 ± 1.4	83.5 ± 2.6	96.6 ± 1.0	86.8 ± 3.0	98.1 ± 0.8	86.5 ± 3.7
5	96.3 ± 2.5	58.4 ± 2.9	41.5 ± 2.8	96.2 ± 1.7	87.5 ± 1.9	97.6 ± 1.1	91.9 ± 2.3	96.9 ± 1.7	88.3 ± 3.8
6	93.7 ± 3.2	42.3 ± 2.3	35.6 ± 5.0	97.5 ± 0.9	93.1 ± 1.8	97.3 ± 0.9	93.4 ± 2.6	97.3 ± 1.0	91.9 ± 2.5
7	88.0 ± 4.5	48.6 ± 2.6	43.7 ± 3.0	97.4 ± 0.7	92.9 ± 2.0	97.1 ± 0.8	93.3 ± 2.4	97.9 ± 0.9	94.2 ± 2.3
8	88.3 ± 6.0	47.4 ± 3.6	36.6 ± 7.0	95.9 ± 1.0	89.9 ± 2.3	94.6 ± 1.8	92.6 ± 2.0	96.2 ± 0.8	90.4 ± 3.0
9	88.0 ± 5.8	35.9 ± 11.8	25.8 ± 10.8	94.5 ± 1.9	88.9 ± 3.0	95.8 ± 1.0	88.7 ± 1.9	95.6 ± 1.4	88.1 ± 2.5
10	91.0 ± 4.0	11.3 ± 4.8	5.2 ± 3.4	72.4 ± 4.4	67.3 ± 3.6	73.1 ± 3.8	65.8 ± 5.0	73.6 ± 5.4	66.3 ± 5.4

Freq	Dgm0		Dgm1		Dgm0 & Dgm1	
	Train	Test	Train	Test	Train	Test
1	8.3 ± 0.5	3.4 ± 1.1	8.1 ± 0.2	3.7 ± 0.5	8.2 ± 0.3	3.5 ± 0.5
2	8.3 ± 0.3	3.4 ± 0.7	8.2 ± 0.5	3.5 ± 1.1	8.6 ± 0.4	3.0 ± 1.0
3	66.5 ± 2.7	31.8 ± 4.8	50.6 ± 2.1	31.1 ± 4.0	80.5 ± 1.3	44.4 ± 4.3
4	46.2 ± 2.5	27.0 ± 3.8	83.1 ± 1.6	63.5 ± 4.6	89.1 ± 1.5	69.0 ± 4.9
5	28.5 ± 1.4	18.9 ± 4.0	75.2 ± 2.6	58.3 ± 4.6	76.8 ± 2.7	58.4 ± 7.9
6	25.4 ± 1.8	19.0 ± 2.4	96.5 ± 1.1	88.7 ± 2.4	96.8 ± 0.7	89.9 ± 1.7
7	19.4 ± 2.6	10.0 ± 3.4	98.2 ± 0.5	93.6 ± 1.9	98.3 ± 0.6	94.1 ± 2.5
8	10.8 ± 2.6	3.6 ± 2.4	91.9 ± 0.9	88.8 ± 2.7	91.9 ± 1.2	89.7 ± 3.3
9	10.6 ± 2.7	4.3 ± 2.2	63.8 ± 2.7	53.3 ± 5.9	64.9 ± 2.3	53.7 ± 3.8
10	9.2 ± 2.3	3.6 ± 1.7	27.0 ± 3.9	16.2 ± 3.2	27.3 ± 3.4	18.6 ± 5.6

Table A.1: Results of classification of shape data as explained in Sec. 4.3.2 using tent functions, with partitioning (top table) and without partitioning (bottom table - copied from [1]). The MSK column gives the original results from [2]. Scores highlighted in green give the best average score across all testing columns; scores highlighted in blue have overlapping intervals of standard deviation with the best score.

Freq	MSK	Dgm0		Dgm1		Dgm0 & Dgm1 (Combined Partitioning)		Dgm0 & Dgm1 (Split Partitioning)	
		Train	Test	Train	Test	Train	Test	Train	Test
1	94.7 ± 5.1	96.6 ± 1.0	75.3 ± 5.2	96.5 ± 2.2	80.6 ± 2.1	98.8 ± 1.2	83.8 ± 2.8	100.0 ± 0.0	90.4 ± 2.4
2	99.3 ± 0.9	94.4 ± 0.6	72.4 ± 4.2	99.2 ± 1.2	86.0 ± 4.4	99.9 ± 0.2	92.2 ± 1.9	100.0 ± 0.0	94.8 ± 1.4
3	96.3 ± 2.2	84.8 ± 1.4	57.4 ± 4.2	98.8 ± 0.7	90.6 ± 2.5	98.7 ± 0.9	91.9 ± 2.7	100.0 ± 0.1	92.0 ± 2.7
4	97.3 ± 1.9	73.8 ± 2.8	44.1 ± 5.5	96.2 ± 1.9	86.0 ± 2.8	96.0 ± 1.9	83.1 ± 3.2	97.9 ± 0.9	82.7 ± 4.3
5	96.3 ± 2.5	66.4 ± 4.7	37.4 ± 4.8	95.8 ± 2.0	89.8 ± 3.1	99.3 ± 0.5	91.0 ± 3.1	96.4 ± 1.2	84.8 ± 3.0
6	93.7 ± 3.2	57.9 ± 4.0	37.0 ± 5.8	97.9 ± 0.7	91.8 ± 2.3	97.8 ± 0.8	89.6 ± 2.3	98.6 ± 0.4	90.5 ± 3.2
7	88.0 ± 4.5	61.9 ± 2.2	39.5 ± 5.4	97.2 ± 1.0	90.5 ± 2.3	97.5 ± 0.8	93.9 ± 1.6	98.3 ± 0.5	92.9 ± 1.7
8	88.3 ± 6.0	69.0 ± 2.9	50.8 ± 4.5	98.2 ± 0.7	91.4 ± 2.0	98.1 ± 0.9	91.0 ± 3.2	99.4 ± 0.5	92.6 ± 1.8
9	88.0 ± 5.8	68.7 ± 6.1	52.5 ± 6.1	94.5 ± 1.5	88.1 ± 2.0	95.2 ± 1.3	86.3 ± 2.7	95.7 ± 2.2	88.9 ± 3.5
10	91.0 ± 4.0	61.5 ± 5.1	46.3 ± 4.0	96.2 ± 1.1	88.7 ± 3.0	96.8 ± 1.2	90.1 ± 2.5	97.9 ± 1.0	88.8 ± 2.6

Freq	Dgm0		Dgm1		Dgm0 & Dgm1	
	Train	Test	Train	Test	Train	Test
1	94.3 ± 0.5	67.1 ± 4.7	99.1 ± 0.3	85.4 ± 3.0	99.8 ± 0.3	90.4 ± 5.3
2	92.1 ± 1.4	60.8 ± 6.3	99.9 ± 0.3	89.9 ± 1.5	100.0 ± 0.0	95.1 ± 2.4
3	83.4 ± 2.4	45.1 ± 2.9	99.6 ± 0.5	88.9 ± 3.0	99.7 ± 0.5	90.0 ± 2.0
4	74.7 ± 2.0	37.4 ± 4.7	99.1 ± 0.7	85.2 ± 2.5	98.6 ± 0.9	84.8 ± 3.9
5	65.3 ± 2.9	27.8 ± 5.0	99.2 ± 0.7	93.0 ± 2.2	99.7 ± 0.4	93.3 ± 2.2
6	67.2 ± 2.5	36.5 ± 3.6	99.2 ± 0.5	93.4 ± 2.8	98.8 ± 0.5	92.9 ± 1.8
7	71.5 ± 2.8	40.9 ± 4.1	98.3 ± 0.7	96.6 ± 0.7	99.0 ± 0.4	95.6 ± 1.4
8	84.2 ± 3.3	63.0 ± 4.5	99.0 ± 0.5	93.0 ± 1.8	99.6 ± 0.4	94.0 ± 2.2
9	83.5 ± 2.7	62.4 ± 5.0	98.4 ± 1.2	92.9 ± 1.5	98.5 ± 1.3	92.6 ± 2.1
10	79.8 ± 2.7	59.0 ± 4.6	96.9 ± 0.6	92.1 ± 1.7	97.7 ± 1.1	89.5 ± 4.6

Table A.2: Results of classification of shape data as explained in Sec. 4.3.2 using interpolating polynomial functions with partitioning (top table) and without partitioning (bottom table - copied from [1]). The MSK column gives the original results from [2]. Scores highlighted in green give the best average score across all testing columns; scores highlighted in blue have overlapping intervals of standard deviation with the best score.

APPENDIX B

TABLES OF RESULTS AND PARAMETERS FROM DIRECTED ORDINAL PARTITION NETWORKS

B.1 Dynamical Systems and Parameters List

List of all dynamical systems and the default parameters from *teaspoon* used in Sec. 5.5. Table B.1 gives the parameter values for the dynamical systems while Table B.2 gives the frequency, length and initial conditions. All systems are defined and default parameters included here originate from a document on the *teaspoon* website [126]. At the time of writing this dissertation, the documentation of the available systems is linked at the top of this page: https://lizliz.github.io/teaspoon/DynSysLib.html.

Specifically, the function in *teaspoon* called *DynamicSystems* is used. The input for *system* is listed in the first column of both tables, the input for *parameters* for that particular system are in Table B.1, and the input for *fs*, *L*, and *InitialConditions* are in Table B.2. These are the default parameters if *dynamic_state* is specified to be *'periodic'* or *'chaotic'*.

B.2 Persistence Scores

Tables B.3, B.4, and B.5 give the calculated scores which are plotted in histograms in Fig. 5.9. Note that since in the unweighted, undirected case all edges are assumed to have weight 1, then all birth and death times are integer valued. Thus, total persistence and maximum persistence are all integer valued.

Dynamical System	***parameters***	
	Periodic	**Chaotic**
base_excited_magnetic_pendulum	[0.1038, 0.208, 9.81, 0.18775, 1.91e-5, 0.022, 3π, 0.003, 1.2, 0.032, 1.257e-6]	[0.1038, 0.208, 9.81, 0.18775, 1.91e-5, 0.021, 3π, 0.003, 1.2, 0.032, 1.257e-6]
burke_shaw_attractor	[12, 4]	[10, 4]
chua	[10.8, 27.0, 1.0,, $-\frac{3}{7}$, $\frac{3}{7}$]	[12.8, 27.0, 1.0, $-\frac{3}{7}$, $\frac{3}{7}$]
coupled_lorenz_rossler	[0.25, $\frac{8}{3}$, 0.2, 5.7, 0.1, 0.1, 0.1, 28, 10]	[0.51, $\frac{8}{3}$, 0.2, 5.7, 0.1, 0.1, 0.1, 28, 10]
coupled_rossler_rossler	[0.25, 0.99, 0.95]	[0.3, 0.99, 0.95]
diffusionless_lorenz_attractor	[0.25]	[0.40]
double_pendulum	[0.4, 0.6, 1, 1]	[1, 0, 0, 0]
double_scroll	[1.0]	[0.8]
driven_pendulum	[1, 9.81, 1, 0.1, 5, 1]	[1, 9.81, 1, 0.1, 5, 2]
driven_van_der_pol_oscillator	[2.9, 5.1.788]	[2, 5, 1.788]
duffing_van_der_pol_oscillator	[0.2, 8, 0.35, 1.3]	[0.2, 8, 0.38, 1.2]
forced_brusselator	[0.4, 1.2, 0.05, 1.1]	[0.4, 1.2, 0.05, 1.0]
hadley_circulation	[0.3, 4, 8, 1]	[0.25, 4, 8, 1]
halvorsens_cyclically_symmetric_attractor	[1.85, 4, 4]	[1.45, 4, 4]
linear_feedback_rigid_body_motion_system	[5.3, −10, −3.8]	[5.0, −10, −3.8]
lorenz	[100, 10, $\frac{8}{3}$]	[105, 10, $\frac{8}{3}$]
moore_spiegel_oscillator	[7.8, 20]	[7.0, 20]
nose_hoover_oscillator	[6]	[1]
rabinovich_frabrikant_attractor	[1.16, 0.87]	[1.13, 0.87]
rayleigh_duffing_oscillator	[0.2, 4, 0.3, 1.4]	[0.2, 4, 0.3, 1.2]
rossler	[0.1, 0.2, 14]	[0.15, 0.2, 14]
rucklidge_attractor	[1.1, 6.7]	[1.6, 6.7]
shaw_van_der_pol_oscillator	[1, 5, 1.4]	[1, 5, 1.8]
simplest_cubic_chaotic_flow	[2.11, 2.5]	[2.05, 2.5]
simplest_piecewise_linear_chaotic_flow	[0.7]	[0.6]
thomas_cyclically_symmetric_attractor	[0.17]	[0.18]
ueda_oscillator	[0.05, 7.5, 1.2]	[0.05, 7.5, 1.0]
WINDMI	[0.9, 2.5]	[0.8, 2.5]

Table B.1: Input for input *parameters* in *DynamicSystems* function in *teaspoon*.

Dynamical System	**Parameters**		
	fs	***L***	***InitialConditions***
base_excited_magnetic_pendulum	200	100	[0.0, 0.0]
burke_shaw_attractor	200	500	[0.6, 0, 0]
chua	50	200	[1.0, 0.0, 0.0]
coupled_lorenz_rossler	50	500	[0.1, 0.1, 0.1, 0, 0, 0]
coupled_rossler_rossler	10	1000	[-0.4, 0.6, 5.8, 0.8, -2, -4]
diffusionless_lorenz_attractor	40	1000	[1, -1, 0.01]
double_pendulum	100	100	*'periodic'*: [0.4, 0.6, 1, 1] *'chaotic'*: [0.0, 3, 0, 0]
double_scroll	20	1000	[0.01, 0.01, 0]
driven_pendulum	50	300	[0, 0]
driven_van_der_pol_oscillator	40	300	[-1.9, 0]
duffing_van_der_pol_oscillator	20	500	[0.2, -0.2]
forced_brusselator	20	500	[0.3, 2]
hadley_circulation	50	500	[-10, 0, 37]
halvorsens_cyclically_symmetric_attractor	200	200	[-5, 0, 0]
linear_feedback_rigid_body_motion_system	100	500	[0.2, 0.2, 0.2]
lorenz	100	100	[10.0⁻10.0, 0.0, 1.0]
moore_spiegel_oscillator	100	500	[0.2, 0.2, 0.2]
nose_hoover_oscillator	20	500	[0, 5, 0]
rabinovich_frabrikant_attractor	30	500	[-1, 0, 0.5]
rayleigh_duffing_oscillator	20	500	[0.3, 0.0]
rossler	15	1000	[-0.4, 0.6, 1]
rucklidge_attractor	50	1000	[1, 0, 4.5]
shaw_van_der_pol_oscillator	25	500	[1.3, 0]
simplest_cubic_chaotic_flow	20	1000	[0, 0.96, 0]
simplest_piecewise_linear_chaotic_flow	40	1000	[0, -0.7, 0]
thomas_cyclically_symmetric_attractor	10	1000	[0.1, 0, 0]
ueda_oscillator	50	500	[2.5, 0.0]
WINDMI	20	1000	[1, 0, 4.5]

Table B.2: Input for inputs *fs*, *L*, and *InitialConditions* in *DynamicSystems* function in *teaspoon*.

Dynamical System	Weighted, Directed		Unweighted, Undirected	
	Periodic	**Chaotic**	**Periodic**	**Chaotic**
base_excited_magnetic_pendulum	17.58	45.73	23.0	66.0
burke_shaw_attractor	5.06	25.58	7.0	41.0
chua	9.67	47.68	12.0	70.0
coupled_lorenz_rossler	7.57	98.1	13.0	155.0
coupled_rossler_rossler	17.51	25.1	22.0	37.0
diffusionless_lorenz_attractor	7.0	20.5	9.0	30.0
double_pendulum	14.83	28.49	20.0	49.0
double_scroll	1.35	14.03	1.0	24.0
driven_pendulum	11.4	37.33	15.0	60.0
driven_van_der_pol_oscillator	2.94	67.33	3.0	103.0
duffing_van_der_pol_oscillator	7.83	66.8	10.0	91.0
forced_brusselator	11.5	68.13	15.0	98.0
hadley_circulation	6.88	89.13	9.0	139.0
halvorsens_cyclically_symmetric_attractor	19.75	60.5	26.0	87.0
linear_feedback_rigid_body_motion_system	3.72	53.04	3.0	80.0
lorenz	10.77	56.84	13.0	87.0
moore_spiegel_oscillator	6.4	11.28	8.0	17.0
nose_hoover_oscillator	5.95	50.8	11.0	77.0
rabinovich_frabrikant_attractor	11.57	83.16	13.0	117.0
rayleigh_duffing_oscillator	5.88	72.33	6.0	107.0
rossler	16.63	34.95	21.0	51.0
rucklidge_attractor	10.32	75.25	13.0	111.0
shaw_van_der_pol_oscillator	7.56	80.42	13.0	113.0
simplest_cubic_chaotic_flow	5.5	27.75	7.0	37.0
simplest_piecewise_linear_chaotic_flow	12.33	28.25	16.0	43.0
thomas_cyclically_symmetric_attractor	11.5	53.71	15.0	86.0
ueda_oscillator	7.65	51.28	9.0	77.0
WINDMI	20.6	30.0	28.0	40.0

Table B.3: Total persistence for all systems with no noise added. Scores are rounded to two decimal places.

Dynamical System	Weighted, Directed		Unweighted, Undirected	
	Periodic	Chaotic	Periodic	Chaotic
base_excited_magnetic_pendulum	10.58	11.33	13.0	15.0
burke_shaw_attractor	4.31	5.0	5.0	6.0
chua	9.67	9.32	12.0	14.0
coupled_lorenz_rossler	3.33	3.35	3.0	3.0
coupled_rossler_rossler	6.96	4.25	10.0	4.0
diffusionless_lorenz_attractor	7.0	10.0	9.0	12.0
double_pendulum	5.02	6.35	5.0	8.0
double_scroll	1.35	6.81	1.0	13.0
driven_pendulum	11.4	7.75	15.0	14.0
driven_van_der_pol_oscillator	2.94	8.67	3.0	14.0
duffing_van_der_pol_oscillator	6.33	8.51	7.0	9.0
forced_brusselator	11.5	9.25	15.0	14.0
hadley_circulation	6.88	7.25	9.0	10.0
halvorsens_cyclically_symmetric_attractor	19.25	8.17	25.0	10.0
linear_feedback_rigid_body_motion_system	3.43	5.78	3.0	6.0
lorenz	10.77	6.99	13.0	10.0
moore_spiegel_oscillator	6.4	7.29	8.0	8.0
nose_hoover_oscillator	3.27	10.0	3.0	17.0
rabinovich_frabrikant_attractor	11.57	5.67	13.0	6.0
rayleigh_duffing_oscillator	5.88	6.17	6.0	7.0
rossler	7.18	4.99	10.0	5.0
rucklidge_attractor	10.32	6.5	13.0	8.0
shaw_van_der_pol_oscillator	4.33	6.83	5.0	9.0
simplest_cubic_chaotic_flow	5.5	14.75	7.0	19.0
simplest_piecewise_linear_chaotic_flow	12.33	8.25	16.0	14.0
thomas_cyclically_symmetric_attractor	11.5	6.16	15.0	11.0
ueda_oscillator	7.54	7.5	9.0	9.0
WINDMI	11.98	12.78	14.0	14.0

Table B.4: Maximum persistence for all systems with no noise added. Scores are rounded to two decimal places.

Dynamical System	Weighted, Directed		Unweighted, Undirected	
	Periodic	Chaotic	Periodic	Chaotic
base_excited_magnetic_pendulum	0.27	0.57	0.26	0.55
burke_shaw_attractor	0.32	0.73	0.41	0.65
chua	0.0	0.6	0.0	0.56
coupled_lorenz_rossler	1.03	1.02	0.9	0.95
coupled_rossler_rossler	0.52	0.89	0.47	0.87
diffusionless_lorenz_attractor	0.0	0.51	0.0	0.5
double_pendulum	0.84	0.96	0.82	0.87
double_scroll	0.0	0.33	0.0	0.3
driven_pendulum	0.0	0.66	0.0	0.57
driven_van_der_pol_oscillator	0.0	0.78	0.0	0.73
duffing_van_der_pol_oscillator	0.34	0.81	0.41	0.79
forced_brusselator	0.0	0.73	0.0	0.68
hadley_circulation	0.0	0.78	0.0	0.71
halvorsens_cyclically_symmetric_attractor	0.04	0.72	0.05	0.68
linear_feedback_rigid_body_motion_system	0.27	0.8	0.0	0.76
lorenz	0.0	0.88	0.0	0.82
moore_spiegel_oscillator	0.0	0.58	0.0	0.65
nose_hoover_oscillator	0.89	0.72	0.88	0.66
rabinovich_frabrikant_attractor	0.0	0.91	0.0	0.87
rayleigh_duffing_oscillator	0.0	0.84	0.0	0.78
rossler	0.49	0.88	0.47	0.82
rucklidge_attractor	0.0	0.73	0.0	0.68
shaw_van_der_pol_oscillator	0.77	0.75	0.76	0.7
simplest_cubic_chaotic_flow	0.0	0.49	0.0	0.48
simplest_piecewise_linear_chaotic_flow	0.0	0.58	0.0	0.53
thomas_cyclically_symmetric_attractor	0.0	0.82	0.0	0.75
ueda_oscillator	0.04	0.84	0.0	0.77
WINDMI	0.47	0.54	0.48	0.57

Table B.5: Persistent entropy for all systems with no noise added. Scores are rounded to two decimal places.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Jose A. Perea, Elizabeth Munch, and Firas A. Khasawneh. Approximating continuous functions on persistence diagrams using template functions. *arXiv:1902.07190*, 2019.

[2] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015. doi: 10.1109/cvpr.2015.7299106.

[3] Nathaniel Saul and Chris Tralie. Scikit-tda: Topological data analysis for python, 2019. URL https://doi.org/10.5281/zenodo.2533369.

[4] Floris Takens. Detecting strange attractors in turbulence. In *Dynamical systems and turbulence, Warwick 1980*, pages 366–381. Springer, 1981.

[5] José A. Perea and John Harer. Sliding windows and persistence: An application of topological methods to signal analysis. *Foundations of Computational Mathematics*, pages 1–40, 2015. ISSN 1615-3375. doi: 10.1007/s10208-014-9206-z.

[6] Nicole Sanderson, Elliott Shugerman, Samantha Molnar, James D Meiss, and Elizabeth Bradley. Computational topology techniques for characterizing time-series data. In *International symposium on intelligent data analysis*, pages 284–296. Springer, 2017.

[7] Joshua R. Tempelman and Firas A. Khasawneh. A look into chaos detection through topological data analysis. *Physica D: Nonlinear Phenomena*, 406:132446, 2020.

[8] Slobodan Maletić, Yi Zhao, and Milan Rajković. Persistent topological features of dynamical systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 26(5):053105, 2016. doi: 10.1063/1.4949472.

[9] Boyan Xu, Christopher J Tralie, Alice Antia, Michael Lin, and Jose A Perea. Twisty takens: A geometric characterization of good observations on dense trajectories. *Journal of Applied and Computational Topology*, 3(4):285–313, 2019.

[10] Hitesh Gakhar and Jose A Perea. Sliding window persistence of quasiperiodic functions. *arXiv:2103.04540*, 2021.

[11] Firas A. Khasawneh and Elizabeth Munch. Chatter detection in turning using persistent homology. *Mechanical Systems and Signal Processing*, 70-71:527–541, 2016. ISSN 0888-3270. doi: 10.1016/j.ymssp.2015.09.046.

[12] Firas A. Khasawneh and Elizabeth Munch. Utilizing topological data analysis for studying signals of time-delay systems. In *Advances in Delays and Dynamics*, pages 93–106. Springer

International Publishing, 2017. doi: 10.1007/978-3-319-53426-8_7.

[13] Firas A. Khasawneh and Elizabeth Munch. Topological data analysis for true step detection in periodic piecewise constant signals. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 474(2218):20180027, 2018. doi: 10.1098/rspa.2018.0027.

[14] Melih C. Yesilli, Sarah Tymochko, Firas A. Khasawneh, and Elizabeth Munch. Chatter diagnosis in milling using supervised learning and topological features vector. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019. doi: 10.1109/icmla.2019.00200.

[15] Jesse Berwald and Marian Gidea. Critical transitions in a model of a genetic regulatory system. *Mathematical Biosciences & Engineering*, 11(4):723–740, 2014.

[16] Marian Gidea. Topological data analysis of critical transitions in financial networks. In Puzis R. Shmueli E., Barzel B., editor, *3rd International Winter School and Conference on Network Science NetSci-X 2017*, Springer Proceedings in Complexity. Springer, Cham, 2017.

[17] Christopher J Tralie and Jose A Perea. (quasi) periodicity quantification in video data, using topology. *SIAM Journal on Imaging Sciences*, 11(2):1049–1077, 2018.

[18] Christopher Tralie. High-dimensional geometry of sliding window embeddings of periodic videos. In *32nd International Symposium on Computational Geometry (SoCG 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[19] Yu-Min Chung, Chuan-Shen Hu, Yu-Lun Lo, and Hau-Tieng Wu. A persistent homology approach to heart rate variability analysis with an application to sleep-wake classification. *Frontiers in physiology*, 12:202, 2021.

[20] Sarah Tymochko, Kritika Singhal, and Giseon Heo. Classifying sleep states using persistent homology and markov chains: A pilot study. In *Advances in Data Science*, pages 253–289. Springer, 2021.

[21] Tegan H. Emerson, Sarah Tymochko, George Stantchev, Jason A. Edelberg, Michael Wilson, and Colin C. Olson. A topological approach for motion track discrimination. *arXiv:2102.05705*, 2021.

[22] Sarah Tymochko, Alexander Soloway, Timothy Doster, Colin C. Olson, and Tegan H. Emerson. I spy in the sky: a stable topological approach for aerial tracking data. *Preprint*, 2021.

[23] Ashleigh Thomas, Kathleen Bates, Alex Elchesen, Iryna Hartsock, Hang Lu, and Peter Bubenik. Topological data analysis of c. elegans locomotion and behavior. *Frontiers in*

Artificial Intelligence, 4, 2021.

[24] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

[25] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[26] Holger Kantz and Thomas Schreiber. *Nonlinear time series analysis*, volume 7. Cambridge university press, 2004.

[27] Sarah Tymochko, Elizabeth Munch, Jason Dunion, Kristen Corbosiero, and Ryan Torn. Using persistent homology to quantify a diurnal cycle in hurricanes. *Pattern Recognition Letters*, 133:137–143, may 2020. doi: 10.1016/j.patrec.2020.02.022.

[28] Sarah Tymochko, Elizabeth Munch, and Firas A. Khasawneh. Adaptive partitioning for template functions on persistence diagrams. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019. doi: 10.1109/icmla.2019.00202.

[29] Sarah Tymochko, Elizabeth Munch, and Firas A. Khasawneh. Using zigzag persistent homology to detect hopf bifurcations in dynamical systems. *Algorithms*, 13(11):278, oct 2020. doi: 10.3390/a13110278.

[30] Katharine Turner, Yuriy Mileyko, Sayan Mukherjee, and John Harer. Fréchet means for distributions of persistence diagrams. *Discrete & Computational Geometry*, 52(1):44–70, 2014.

[31] Firas A. Khasawneh, Elizabeth Munch, and Jose A. Perea. Chatter classification in turning using machine learning and topological data analysis. *IFAC-PapersOnLine*, 51(14):195–200, 2018. doi: 10.1016/j.ifacol.2018.07.222.

[32] Paul Bendich, James S Marron, Ezra Miller, Alex Pieloch, and Sean Skwerer. Persistent homology analysis of brain artery trees. *The annals of applied statistics*, 10(1):198, 2016.

[33] Henry Adams and Michael Moy. Topology applied to machine learning: From global to local. *Frontiers in Artificial Intelligence*, 4, 2021. ISSN 2624-8212. doi: 10.3389/frai.2021.668302. URL https://www.frontiersin.org/article/10.3389/frai.2021.668302.

[34] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & computational geometry*, 37(1):103–120, 2007.

[35] Primoz Skraba and Katharine Turner. Wasserstein stability for persistence diagrams. *arXiv:2006.16824*, 2020.

[36] Hassler Whitney. Differentiable manifolds. *Annals of Mathematics*, pages 645–680, 1936.

[37] Andrew M. Fraser and Harry L. Swinney. Independent coordinates for strange attractors from mutual information. *Physical Review A*, 33(2):1134–1140, 1986. doi: 10.1103/physreva.33.1134.

[38] Matthew B. Kennel, Reggie Brown, and Henry D. I. Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys. Rev. A*, 45:3403–3411, 1992. doi: 10.1103/PhysRevA.45.3403.

[39] Louis M. Pecora, Linda Moniz, Jonathan Nichols, and Thomas L. Carroll. A unified approach to attractor reconstruction. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 17 (1):013110, mar 2007. doi: 10.1063/1.2430294.

[40] David Chelidze. Reliable estimation of minimum embedding dimension through statistical analysis of nearest neighbors. *Journal of Computational and Nonlinear Dynamics*, 12(5), jul 2017. doi: 10.1115/1.4036814.

[41] Shaowu Pan and Karthik Duraisamy. On the structure of time-delay embedding in linear models of non-linear dynamical systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(7):073135, jul 2020. doi: 10.1063/5.0010886.

[42] H.S. Kim, R. Eykholt, and J.D. Salas. Nonlinear dynamics, delay times, and embedding windows. *Physica D: Nonlinear Phenomena*, 127(1-2):48–60, mar 1999. doi: 10.1016/s0167-2789(98)00240-1.

[43] Audun Myers and Firas A Khasawneh. On the automatic parameter selection for permutation entropy. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(3):033130, 2020.

[44] Jason P Dunion, Christopher D Thorncroft, and Christopher S Velden. The tropical cyclone diurnal cycle of mature hurricanes. *Monthly Weather Review*, 142(10):3900–3919, 2014.

[45] Tomasz Kaczynski, Konstantin Michael Mischaikow, and Marian Mrozek. *Computational homology*, volume 3. Springer, 2004.

[46] Kenneth R. Knapp and Scott L. Wilkins. Gridded satellite (GridSat) GOES and CONUS data. *Earth System Science Data*, 10(3):1417–1425, 2018. doi: 10.5194/essd-10-1417-2018.

[47] Azriel Rosenfeld and John L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13(4):471–494, October 1966. ISSN 0004-5411. doi: 10.1145/321356.321357.

[48] Azriel Rosenfeld and John L. Pfaltz. Distance functions on digital pictures. *Pattern Recognition*, 1(1):33 – 61, 1968. ISSN 0031-3203. doi: 10.1016/0031-3203(68)90013-7.

[49] Konstantin Mischaikow and Vidit Nanda. Morse theory for filtrations and efficient computation of persistent homology. *Discrete & Computational Geometry*, 50(2):330–353, 2013.

[50] Vidit Nanda. Perseus. http://www.sas.upenn.edu/~vnanda/perseus/, 2015.

[51] Sarah D. Ditchek, Kristen L. Corbosiero, Robert G. Fovell, and John Molinari. Electrically active tropical cyclone diurnal pulses in the atlantic basin. *Monthly Weather Review*, 147 (10):3595–3607, 2019. doi: 10.1175/MWR-D-19-0129.1.

[52] Vanessa Robins, Peter J Wood, and Adrian P Sheppard. Theory and algorithms for constructing discrete Morse complexes from grayscale digital images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:1646–1658, 05 2011. doi: 10.1109/TPAMI.2011.95.

[53] AL Herring, Vanessa Robins, and AP Sheppard. Topological persistence for relating microstructure and capillary fluid trapping in sandstones. *Water Resources Research*, 55(1): 555–573, 2019.

[54] O. Delgado-Friedrichs, V. Robins, and A. Sheppard. Morse theory and persistent homology for topological analysis of 3D images of complex materials. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 4872–4876, October 2014. doi: 10.1109/ICIP.2014.7025987.

[55] Chul Moon, Scott A Mitchell, Jason E Heath, and Matthew Andrew. Statistical inference over persistent homology predicts fluid flow in porous media. *Water Resources Research*, 55(11):9592–9603, 2019.

[56] Chuan-Shen Hu, Austin Lawson, Yu-Min Chung, and Kaitlin Keegan. Two-parameter persistence for images via distance transform. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4176–4184, 2021.

[57] Teresa Heiss, Sarah Tymochko, Brittany Story, Adélie Garin, Hoa Bui, Bea Bleile, and Vanessa Robins. The impact of changes in resolution on the persistent homology of images. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 3824–3834, 2021. doi: 10.1109/BigData52589.2021.9671483.

[58] E. Oran Brigham. *The Fast Fourier Transform and Its Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. ISBN 0-13-307505-2.

[59] J. Serra. *Image Analysis and Mathematical Morphology*. Number v. 1 in Image Analysis and Mathematical Morphology. Academic Press, 1984. ISBN 9780126372427.

[60] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[61] Guillem Quintana and Joaquim Ciurana. Chatter in machining processes: A review. *International Journal of Machine Tools and Manufacture*, 51(5):363–376, 2011.

[62] Yuriy Mileyko, Sayan Mukherjee, and John Harer. Probability measures on the space of persistence diagrams. *Inverse Problems*, 27(12):124007, 2011.

[63] Elizabeth Munch, Katharine Turner, Paul Bendich, Sayan Mukherjee, Jonathan Mattingly, and John Harer. Probabilistic fréchet means for time varying persistence diagrams. *Electronic Journal of Statistics*, 9(1):1173–1204, 2015.

[64] Alexander Wagner. Nonembeddability of persistence diagrams with $p > 2$ wasserstein metric. *Proceedings of the American Mathematical Society*, 149(6):2673–2677, 2021.

[65] Peter Bubenik and Alexander Wagner. Embeddings of persistence diagrams into hilbert spaces. *Journal of Applied and Computational Topology*, 4(3):339–351, 2020.

[66] Mathieu Carriere, Marco Cuturi, and Steve Oudot. Sliced wasserstein kernel for persistence diagrams. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 664–673. JMLR. org, 2017.

[67] Genki Kusano, Kenji Fukumizu, and Yasuaki Hiraoka. Persistence weighted gaussian kernel for topological data analysis. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 2004–2013. JMLR.org, 2016. URL http://dl.acm.org/citation.cfm?id=3045390.3045602.

[68] Roland Kwitt, Stefan Huber, Marc Niethammer, Weili Lin, and Ulrich Bauer. Statistical topological data analysis - a kernel perspective. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NeurIPS'15, pages 3070–3078, Cambridge, MA, USA, 2015. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969442.2969582.

[69] Tam Le and Makoto Yamada. Persistence fisher kernel: A riemannian manifold kernel for persistence diagrams. In *NeurIPS*, 2018.

[70] Tullia Padellini and Pierpaolo Brutti. Supervised learning with indefinite topological kernels. *Statistics*, pages 1–22, 2021.

[71] Xiaojin Zhu, Ara Vartanian, Manish Bansal, Duy Nguyen, and Luke Brandl. Stochastic multiresolution persistent homology kernel. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 2449–2455. AAAI Press, 2016. ISBN 978-1-57735-770-4. URL http://dl.acm.org/citation.cfm?id=3060832.3060964.

[72] Qi Zhao and Yusu Wang. Learning metrics for persistence-based summaries and applications for graph classification. In *Advances in Neural Information Processing Systems*, pages 9855–9866, 2019.

[73] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *The Journal of Machine Learning Research*, 18(1):218–252, January 2017. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=3122009.3122017.

[74] Eric Berry, Yen-Chi Chen, Jessi Cisewski-Kehe, and Brittany Terese Fasy. Functional summaries of persistence diagrams. *Journal of Applied and Computational Topology*, 4(2): 211–262, 2020.

[75] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16:77–102, 2015. URL http://jmlr.org/papers/v16/bubenik15a.html.

[76] Peter Bubenik and Paweł Dłotko. A persistence landscapes toolbox for topological statistics. *Journal of Symbolic Computation*, 78:91–114, 2017. doi: 10.1016/j.jsc.2016.03.009.

[77] Gunnar Carlsson, Tigran Ishkhanov, Vin de Silva, and Afra Zomorodian. On the local behavior of spaces of natural images. *International Journal of Computer Vision*, 76:1–12, 2008. ISSN 0920-5691. doi: 10.1007/s11263-007-0056-x.

[78] Mathieu Carrière, Steve Y. Oudot, and Maks Ovsjanikov. Stable topological signatures for points on 3d shapes. *Computer Graphics Forum*, 34(5):1–12, 2015. doi: 10.1111/cgf.12692.

[79] Ilya Chevyrev, Vidit Nanda, and Harald Oberhauser. Persistence paths and signature features in topological data analysis. *IEEE transactions on pattern analysis and machine intelligence*, 42(1):192–202, 2018.

[80] David Rouse, Adam Watkins, David Porter, John Harer, Paul Bendich, Nate Strawn, Elizabeth Munch, Jonathan DeSena, Jesse Clarke, Jeffrey Gilbert, Peter Chin, and Andrew Newman. Feature-aided multiple hypothesis tracking using topological and statistical behavior classifiers. In Ivan Kadar, editor, *Signal Processing, Sensor/Information Fusion, and Target Recognition XXIV*. SPIE, 2015. doi: 10.1117/12.2179555.

[81] Bartosz Zieliński, Michał Lipiński, Mateusz Juda, Matthias Zeppelzauer, and Paweł Dłotko. Persistence codebooks for topological data analysis. *Artificial Intelligence Review*, 54(3): 1969–2009, 2021.

[82] Chi Seng Pun, Si Xian Lee, and Kelin Xia. Persistent-homology-based machine learning: a survey and a comparative study. *Artificial Intelligence Review*, pages 1–45, 2022.

[83] Danielle Barnes, Luis Polanco, and Jose A Perea. A comparative study of machine learning methods for persistence diagrams. *Frontiers in Artificial Intelligence*, 4, 2021.

[84] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[85] Audun D Myers, Melih Yesilli, Sarah Tymochko, Firas Khasawneh, and Elizabeth Munch. Teaspoon: A comprehensive python package for topological signal processing. In *NeurIPS 2020 Workshop on Topological Data Analysis and Beyond*, 2020. URL https://openreview.net/forum?id=qUoVqrIcy2P.

[86] David Pickup, Xianfang Sun, Paul L Rosin, Ralph R Martin, Z Cheng, Zhouhui Lian, Masaki Aono, A Ben Hamza, A Bronstein, M Bronstein, et al. Shape retrieval of non-rigid 3d human models. *International Journal of Computer Vision*, 120(2):169–193, 2016.

[87] Michael McCullough, Michael Small, Thomas Stemler, and Herbert Ho-Ching Iu. Time lagged ordinal partition networks for capturing dynamics of continuous dynamical systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 25(5):053101, may 2015. doi: 10.1063/1.4919075.

[88] Steven H. Strogatz. *Nonlinear Dynamics and Chaos*. Taylor & Francis Inc, 2014. ISBN 0813349109. URL https://www.ebook.de/de/product/20639922/steven_h_strogatz_nonlinear_dynamics_and_chaos.html.

[89] Alexander Khor and Michael Small. Examining k-nearest neighbour networks: Superfamily phenomena and inversion. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 26 (4):043101, apr 2016. doi: 10.1063/1.4945008.

[90] Lucas Lacasa, Bartolo Luque, Fernando Ballesteros, Jordi Luque, and Juan Carlos Nuno. From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*, 105(13):4972–4975, 2008.

[91] Bartolo Luque, Lucas Lacasa, Fernando Ballesteros, and Jordi Luque. Horizontal visibility graphs: Exact results for random time series. *Physical Review E*, 80(4):046103, 2009.

[92] N Marwan, M Carmenromano, M Thiel, and J Kurths. Recurrence plots for the analysis of complex systems. *Physics Reports*, 438(5-6):237–329, jan 2007. doi: 10.1016/j.physrep.2006.11.001.

[93] Zhongke Gao and Ningde Jin. Complex network from time series based on phase space reconstruction. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 19(3):033137, 2009.

[94] J.-P Eckmann, S. Oliffson Kamphorst, and D Ruelle. Recurrence plots of dynamical systems. *Europhysics Letters (EPL)*, 4(9):973–977, nov 1987. doi: 10.1209/0295-5075/4/9/004.

[95] 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, 2011.

[96] Michael Small. Complex networks from time series: Capturing dynamics. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*. IEEE, may 2013. doi: 10.1109/iscas.2013.6572389.

[97] Christopher W Kulp, Jeremy M Chobot, Helena R Freitas, and Gene D Sprechini. Using

ordinal partition transition networks to analyze ECG data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 26(7):073114, 2016.

[98] Jiayang Zhang, Jie Zhou, Ming Tang, Heng Guo, Michael Small, and Yong Zou. Constructing ordinal partition transition networks from multivariate time series. *Scientific reports*, 7(1): 1–13, 2017.

[99] Michael McCullough, Konstantinos Sakellariou, Thomas Stemler, and Michael Small. Regenerating time series from ordinal networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(3):035814, 2017.

[100] Christoph Bandt and Bernd Pompe. Permutation entropy: a natural complexity measure for time series. *Physical review letters*, 88(17):174102, 2002.

[101] Audun Myers, Elizabeth Munch, and Firas A. Khasawneh. Persistent homology of complex networks for dynamic state detection. *Physical Review E*, 100(2), 2019. doi: 10.1103/physreve.100.022314.

[102] Paul A Gagniuc. *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017.

[103] Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

[104] Samir Chowdhury and Facundo Mémoli. A functorial dowker theorem and persistent homology of asymmetric networks. *Journal of Applied and Computational Topology*, 2(1): 115–175, 2018.

[105] Harish Chintakunta, Thanos Gentimis, Rocio Gonzalez-Diaz, Maria-Jose Jimenez, and Hamid Krim. An entropy-based persistence barcode. *Pattern Recognition*, 48(2):391–401, 2015.

[106] Nieves Atienza, Rocio Gonzalez-Díaz, and Manuel Soriano-Trigueros. On the stability of persistent entropy and new summary functions for topological data analysis. *Pattern Recognition*, 107:107509, 2020. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2020.107509. URL https://www.sciencedirect.com/science/article/pii/S0031320320303125.

[107] Samir Chowdhury and Facundo Mémoli. Persistent path homology of directed networks. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1152–1169. SIAM, 2018.

[108] Pawe Dotko, Kathryn Hess, Ran Levi, Max Nolte, Michael Reimann, Martina Scolamiero, Katharine Turner, Eilif Muller, and Henry Markram. Topological analysis of the connectome of digital reconstructions of neural microcircuits. *arXiv:1601.01580*, 2016.

[109] Michael W Reimann, Max Nolte, Martina Scolamiero, Katharine Turner, Rodrigo Perin, Giuseppe Chindemi, Paweł Dłotko, Ran Levi, Kathryn Hess, and Henry Markram. Cliques of neurons bound into cavities provide a missing link between structure and function. *Frontiers in computational neuroscience*, 11:48, 2017.

[110] Daniel Lütgehetmann, Dejan Govc, Jason P Smith, and Ran Levi. Computing persistent homology of directed flag complexes. *Algorithms*, 13(1):19, 2020.

[111] Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Anibal M Medina-Mardones, Alberto Dassatti, and Kathryn Hess. giotto-tda:: A topological data analysis toolkit for machine learning and data exploration. *J. Mach. Learn. Res.*, 22: 39–1, 2021.

[112] Katharine Turner. Rips filtrations for quasimetric spaces and asymmetric functions with stability results. *Algebraic & Geometric Topology*, 19(3):1135–1170, 2019.

[113] Gunnar Carlsson and Vin de Silva. Zigzag persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010. doi: 10.1007/s10208-010-9066-0.

[114] Peter Gabriel. Unzerlegbare darstellungen i. *Manuscripta mathematica*, 6(1):71–103, 1972.

[115] Harm Derksen and Jerzy Weyman. Quiver representations. *Notices of the AMS*, 52(2): 200–206, 2005.

[116] John Guckenheimer and Philip Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*. Springer New York, 1983. doi: 10.1007/978-1-4612-1140-2.

[117] Jesse Berwald, Marian Gidea, and Mikael Vejdemo-Johansson. Automatic recognition and tagging of topologically different regimes in dynamical systems. *arXiv:1312.2482*, 2013.

[118] Marian Gidea and Yuri Katz. Topological data analysis of financial time series: Landscapes of crashes. *Physica A: Statistical Mechanics and its Applications*, 491:820–834, 2018.

[119] Marian Gidea, Daniel Goldsmith, Yuri Katz, Pablo Roldan, and Yonah Shmalo. Topological recognition of critical transitions in time series of cryptocurrencies. *Physica A: Statistical Mechanics and its Applications*, page 123843, 2020.

[120] Mikael Vejdemo-Johansson, Florian T Pokorny, Primoz Skraba, and Danica Kragic. Cohomological learning of periodic motion. *Applicable algebra in engineering, communication and computing*, 26(1-2):5–26, 2015.

[121] Gunnar Carlsson, Vin De Silva, and Dmitriy Morozov. Zigzag persistent homology and real-valued functions. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 247–256, 2009.

[122] EE Sel'Kov. Self-oscillations in glycolysis. *European Journal of Biochemistry*, 4(1):79–86, 1968.

[123] Nicholas J Cavanna, Mahmoodreza Jahanseir, and Donald R Sheehy. A geometric perspective on sparse filtrations. *arXiv:1506.03797*, 2015.

[124] Gunnar Carlsson and Afra Zomorodian. The theory of multidimensional persistence. *Discrete & Computational Geometry*, 42(1):71–93, 2009.

[125] The RIVET Developers. Rivet. https://github.com/rivetTDA/rivet/, 2020.

[126] Audun Myers. Dynamic systems library documentation. https://lizliz.github.io/teaspoon/DynSysLib.html, 2020.