# Teaspoon: A comprehensive python package for topological signal processing

**Audun D. Myers**
Dept. of Mechanical Engineering
Michigan State University
East Lansing, MI 48824
myersau3@msu.edu

**Melih C. Yesilli**
Dept. of Mechanical Engineering
Michigan State University
East Lansing, MI 48824
yesillim@egr.msu.edu

**Sarah Tymochko**
Dept. of Computational Mathematics,
Science and Engineering
Michigan State University
East Lansing, MI 48824
tymochko@egr.msu.edu

**Firas Khasawneh**
Dept. of Mechanical Engineering
Michigan State University
East Lansing, MI 48824
khasawn3@msu.edu

**Elizabeth Munch**
Dept. of Computational Mathematics,
Science and Engineering
Michigan State University
East Lansing, MI 48824
muncheli@msu.edu

## Abstract

The emerging field of *topological signal processing* brings methods from Topological Data Analysis (TDA) to create new tools for signal processing by incorporating aspects of shape. In this paper, we present an overview of the python package `teaspoon` , which brings together available software for computing persistent homology, the main workhorse of TDA, with modules that expand the functionality of `teaspoon` as a state-of-the-art topological signal processing tool. These modules include methods for incorporating tools from machine learning, complex networks, information, and parameter selection along with a dynamical systems library to streamline the creation and benchmarking of new methods. All code is open source with up to date documentation, making the code easy to use, in particular for signal processing experts with limited experience in topological methods.

## 1 Introduction

Topological signal processing is a newly emerging field with an ever growing collection of tools. Using Topological Data Analysis (TDA) for signal processing allows for an analysis of the underlying shape of a time series. These methods are well backed by theory [1, 2] and have shown success in numerous application areas including machining dynamics [3, 4, 5, 6, 7], finance [8, 9], and gene expression [10, 11].

Here we present the python package, `teaspoon`, that provides state-of-the-art topological signal processing tools as well as wrappers for available persistent homology software. While some

the Topological Data Analysis and Beyond Workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

TDA based packages exist for python (e.g. Scikit-TDA and Giotto-TDA), the `teaspoon` package specifically provides modules design to tackle questions related to signal processing and time series analysis from the viewpoint of topology. In comparison, other existing packages are designed for more general applications for TDA.

In the `teaspoon` package there are currently five main modules: dynamical systems, machine learning, complex networks, information, and parameter selection with several sub-modules for each as shown in Fig. 1. The dynamical systems library is currently hosting 60 dynamical systems including maps, flows, and collected data sets. The machine learning library contains code for numerous persistence diagram featuriztion and kernel methods. Specifically, this module includes the template function featurization methods described in [12, 13] as well as persistence landscapes [14], persistence images [15], Carlsson coordinates [16], persistence paths and signature [17, 18] and the multi-scale kernel method [19]. The complex networks module contains code to represent a time series as a network using ordinal partitions [20] or $k$ nearest neighbors [21]. This module also provides several methods for calculating distances between nodes based on the adjacency matrix, which allows for the calculation of the persistent homology of the resulting networks. The information theory module implements entropy based functions for signal processing persistence diagram analysis. Lastly, the parameter selection module currently provides multiple algorithms for the automatic selection of the delay $\tau$ and dimension $n$ parameters for state space reconstruction and permutation entropy.

In this work, we outline the features available in each module as well as features that will be added in the future. The goal of this package is to provide a range of topological signal processing tools in one unified framework. Additionally, for most of these modules, further documentation and examples of the functions are provided in the `teaspoon` documentation webpage[1].

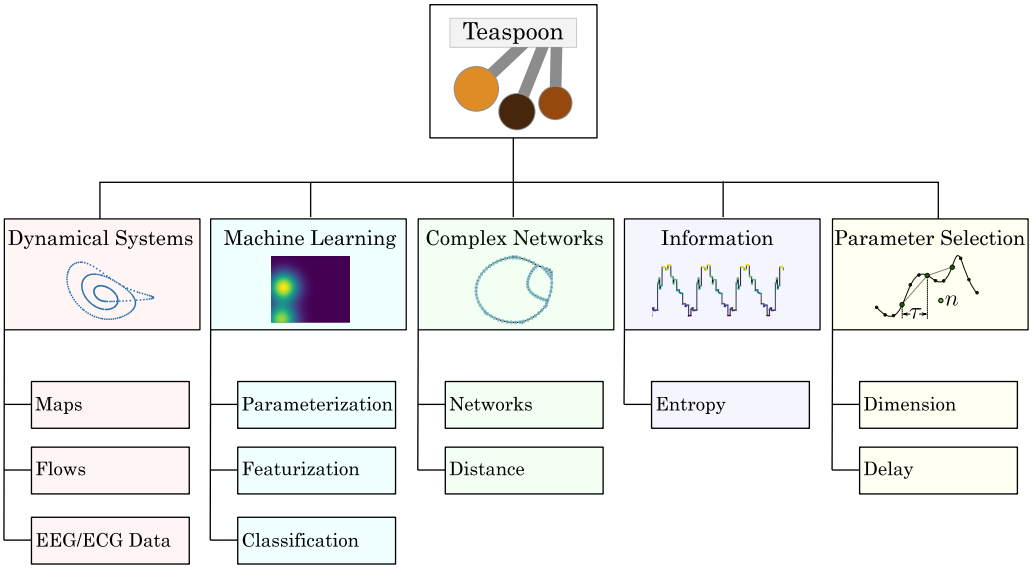


Figure 1: Tree structure of `teaspoon`.

## 2 Dynamical Systems Library (DynSysLib)

The dynamical systems library (DynSysLib) is a `teaspoon` module that provides a wide selection of dynamical system simulation models with many from [22]. Most of the available dynamical systems are able to exhibit both periodic and chaotic responses. In general, these systems can be separated into three categories: (1) flows, (2) Maps, and (3) Collected data. A full list of the available dynamical systems are provided in tables 1 and 2 of the appendix. The module has a single function *DynamicSystems*, which allows a wide range of the simulation control with the user being able to control as little as the system of interest and the desired dynamical state (chaotic or periodic) or provide detailed simulation parameters such as initial conditions, system parameters, and solution

time. The function output is the resulting time series response. For details on the default parameters used, equations of motion, and examples, please see the `teaspoon` documentation webpage[1].

# 3 Machine Learning Module

In this section, we describe the machine learning module in `teaspoon`. Machine learning module provides automated feature matrix generation and classification, and it is suitable for the applications where persistence diagrams can be computed. There are three main files inside the module and these are $Base.py$, $feature\_functions.py$, and $PD\_Classification.py$. Here, we will explain the necessary functions in each of these files and show how to use these functions to perform machine learning using Topological Data Analysis (TDA).

## 3.1 Parameter Buckets

The parameter bucket is a tool to hold all necessary parameters for the featurization functions as well as the classification algorithms. This includes parameters such as the classification algorithm, the size of the test set, as well as the desired persistence diagram featurization method. The parameter buckets are implemented as classes in the $Base.py$ file. The basic structure is implemented as a class $ParameterBucket$, however there are two more specialized classes, $InterPolyParameters$ and $TentParameters$ that are dedicated for parameters to the template functions introduced in Ref.[12]. These parameter buckets also have the functionality to use the template function featurization on localized regions of the persistence diagrams, using an adaptive partitioning method described in Ref.[13].

The rest of the parameter buckets are used for other featurization methods. The $LandscapesParameterBucket$ is for persistence landscapes [14], which requires an input for the landscape number that will be used to generate feature matrix. The $CL\_ParameterBucket$ is used to set parameters for classification using Persistence Images [15], Carlsoon Coordinates [16], persistence paths and signature [17, 18] and kernel method [19].

## 3.2 Featurization

The file $feature\_functions.py$ contains functions that compute the topological features mentioned above. $F\_$ and $CL\_$ suffixes indicate that corresponding functions are designed for featurization and classification, respectively. First, for the template featurizations, there are two main functions, $tents$ and $interp\_polynomial$. These functions compute the collection of template functions based on a grid formed using parameters from the corresponding parameter buckets.

In addition to these, there is $PLandscape$ class that uses $PLandscapes$ function to compute the persistence landscapes for a given persistence diagram [14]. This class has an option to define $L\_number$ which returns specific landscapes in an array. Output of $PLandscapes$ is a dictionary that includes all landscapes, total number of landscapes and the desired landscapes if user defines $L\_number$. $PLandscape$ class can also plot persistence landscapes. If user does not define the desired landscapes to plot, all landscapes will be plotted. $F\_Landscape$ uses persistence landscapes to compute feature matrix as explained in Ref. [23]. The inputs of the function are persistence landscapes, parameter bucket object that is explained in Sec. 3.1.

The second featurization method is persistence images. We utilized Persistence Images package to compute persistence images. $F\_Image$ takes persistence diagrams, pixel size, variance of the Gaussian distribution, the numbers of persistence diagrams whose image will be plotted, and transfer learning option. If transfer learning option is set to true, second set of persistence diagrams should be provided. Then, it will compute feature matrices for both sets of diagrams. Carlsson Coordinates is the third featurization method [16]. It has five coordinates that depend on birth and death times of persistence diagrams. $F\_CCoordinates$ takes persistence diagrams and computes these five features. It has second input, $FN$ that defines how many feature will be computed. Feature vectors are generated using $\sum_{i=1}^{FN} \binom{FN}{i}$ combinations of five coordinates. $F\_CCoordinates$ will return these feature vectors, number of combinations and combinations in a list.

[1] http://elizabethmunch.com/code/teaspoon

Another featurization method is persistence path and signatures [17, 18]. $F\_PSignature$ function computes signatures on persistence landscapes. The first two levels of signatures are currently coded in the function. The inputs are persistence landscapes and the number of the landscape which will be used to compute the signatures. Then it returns the feature matrix to be used in the classification. Final featurization method is kernel method for persistence diagrams. $KernelMethod$ computes the kernel between given two persistence diagrams. It also has $sigma$ input which is a variable in the formula of the kernel given in Ref. [19]. After computing pairwise kernels between the diagrams, it can be used as pre-defined kernel in Support Vector Machine (SVM) algorithm for classification.

### 3.3 Classification

Classification functions are embedded in $PD\_Classification$. Most of the functions take feature functions and parameter bucket object as input. They divide the given feature matrix into training set and test set with respect to test size defined in the parameter bucket. Classification can be performed using four classification algorithms: Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RF) and Gradient Boosting (GB). For the kernel method, LibSVM package [24] is utilized to insert pre-computed kernel matrix for classification. Additionally, the featurization methods can be used to create feature vectors compatible with any scikit-learn classification algorithm.

We also include the option of Transfer learning in classification for most of the featurization methods except kernel method. In this type of classification, a classifier is trained on a data set and tested on another one. One can refer to Ref. [25] for more details about transfer learning. When user defines the transfer learning as true in parameter bucket, feature functions will be computed for training and test persistence diagrams separately. In both classification type, training and test set will be generated 10 times randomly. Mean classification score, standard deviation for training and test set and total runtime for the classification are given as output.

## 4 Complex Networks Module

The `teaspoon` module provides the Python implementation of the algorithms used in [26], which provides methods for analyzing the dynamic state of a time series based on the persistent homology of the network representations of time series. The general pipeline, as shown in Fig. 2, is as follows: (1) represent a time series as a network as described in Section 4.1, (2) Generate a distance matrix from the undirected and weighted adjacency matrix as described in section 4.2, and (3) apply 1-D persistent homology to the distance matrix. The persistence diagram point summaries can be generated to analyze the dynamic state of the underlying time series.

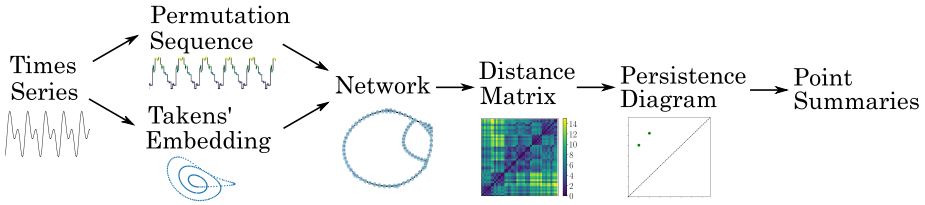


Figure 2: The persistent homology of complex networks pipeline.

### 4.1 Network Representations of Time Series

There are currently two available algorithms in the complex networks module to represent a time series as a complex network. Specifically, these are $k$ Nearest Neighbor ($k$-NN) networks [21] and ordinal partition networks [20]. For our implementation of these algorithms we use the adjacency matrix as our graph data structure.

For the ordinal partition network a permutation sequence needs to be generated by using the function *permutation_sequence*, which requires a time series and the permutation dimension $n$ and delay $\tau$. For selecting the dimension and delay we suggest using the parameter selection module. Using the permutation sequence, the resulting adjacency matrix is formed using the *AdjacenyMatrix_OP* function, which creates edges in the graph based on permutation transitions.

Two steps are required to generate $k$-NN networks. First, the time series needs to have its state space reconstructed through Takens' embedding, which is done through the function *Takens_Embedding*. This function requires the time series and the embedding dimension and delay. The dimension and delay can be selected using the parameter selection module. Next, the $k$-NN are found using the *k_NN* function and specifying $k$ which has a default of $k = 4$. Using the list of neighbors, an adjacency matrix is formed using the *Adjacency_KNN* function by treating each embedded vector as a node and adding edges when two nodes are $k$-NN.

The next step in the pipeline is to define algorithms to represent distances between nodes in the network based on the adjacency matrix, which is discussed in the subsequent section.

### 4.2 Distance Matrix

Two steps are required to assign distances between nodes in a network: (1) apply an edge weight algorithm to represent distances for adjacent nodes and (2) implement a distance algorithm for non-adjacent nodes.

For the first step we provide the following edge weight functions: unweighted, inverse, and difference. Specifically, the unweighted option changes all the edge weights to 1, the inverse sets the weight to the element-wise reciprocal, and the difference finds the maximum edge weight and sets the new edge weight as the difference between the max edge weight and that edge's weight.

The second step requires a method for defining distances between non-adjacent nodes. To do this we offer two options: the shortest-path distance and effective network resistance [27]. Both of these steps are implemented through the *DistanceMatrix* function.

## 5 Information Module

The information theory module currently provides three functions for information entropy calculations. The first two are the calculation of the permutation entropy [28] and multi-scale permutation entropy as *PE* and *MsPE*, respectively. Permutation entropy has been shown to be a useful tool for analyzing signal complexity and has very few requirments for its application. The third function is the persistent entropy [29] through the function *PersistentEntropy*, which calculates the entropy of a persistence diagram given the lifetimes from the persistence diagram.

## 6 Parameter Selection Module

The parameter selection module provides code for the functions used in [30] and [31] for automatically calculating the dimension $n$ and delay $\tau$ parameters for both permutation entropy and Takens' embedding (state space reconstruction). For details on each of the methods please reference their respective publications as some are more suitable for non-linear time series or have time series requirements. A comprehensive list of the available methods are provided in Table 3.

## 7 Future Work

In the future, we would like to include two new modules as well as improve the complex networks module by adding a de-noising procedure for ordinal partition networks. The first new module would focus on sublevel set persistence, with code to quickly compute sublevel set persistence on 1-D functions as well as a statistical analysis function for providing confidence intervals for points in the persistence diagram. The second module is for quick persistence diagram computation, which will include methods such as the Bézier curve approximation, point cloud clustering, and boot strapping point clouds and persistence diagrams.

## References


[1] Jose A Perea and John Harer. Sliding windows and persistence: An application of topological methods to signal analysis. *Foundations of Computational Mathematics*, 15(3):799–838, 2015.

[2] Michael Robinson. *Topological signal processing*. Springer, 2014.

[3] 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.

[4] 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.

[5] Firas A. Khasawneh and Elizabeth Munch. Chatter detection in turning using persistent homology. *Mechanical Systems and Signal Processing*, 70-71:527–541, 2016.

[6] Firas A. Khasawneh, Elizabeth Munch, and Jose A. Perea. Chatter classification in turning using machine learning and topological data analysis. In Tamas Insperger, editor, *14th IFAC Workshop on Time Delay Systems TDS 2018: Budapest, Hungary, 28–30 June 2018*, volume 51, pages 195–200, 2018.

[7] Firas A. Khasawneh and Elizabeth Munch. Stability determination in turning using persistent homology and time series analysis. In *Volume 4B: Dynamics, Vibration, and Control*. American Society of Mechanical Engineers, nov 2014.

[8] 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.

[9] 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.

[10] Jose A. Perea, Anastasia Deckard, Steve B. Haase, and John Harer. SW1pers: Sliding windows and 1-persistence scoring; discovering periodicity in gene expression time series data. *BMC Bioinformatics*, 16(1), 2015.

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

[12] Jose A. Perea, Elizabeth Munch, and Firas A. Khasawneh. Approximating Continuous Functions On Persistence Diagrams Using Template Functions. *arXiv preprint:1902.07190*, 2019.

[13] 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, dec 2019.

[14] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16(3):77–102, 2015.

[15] 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. *J. Mach. Learn. Res.*, 18(1):218–252, January 2017.

[16] Aaron Adcock, Erik Carlsson, and Gunnar Carlsson. The ring of algebraic functions on persistence bar codes. *Homology, Homotopy and Applications*, 18(1):381–402, 2016.

[17] Ilya Chevyrev and Andrey Kormilitzin. A Primer on the Signature Method in Machine Learning. 2016.

[18] Ilya Chevyrev, Vidit Nanda, and Harald Oberhauser. Persistence paths and signature features in topological data analysis. 2018.

[19] 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, jun 2015.

[20] 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.

[21] 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.

[22] J. Sprott. Chaos and time-series analysis. *Choice Reviews Online*, 41(06):41–3492–41–3492, feb 2004.

[23] Melih C. Yesilli, Firas A. Khasawneh, and Andreas Otto. Topological feature vectors for chatter detection in turning processes. *arXiv preprint: 1905.08671*, 2019.

[24] Chih-Chung Chang and Chih-Jen Lin. LIBSVM. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, apr 2011.

[25] Sinno Jialin Pan and Qiang Yang. A Survey On Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, oct 2010.

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

[27] W. Ellens, F.M. Spieksma, P. Van Mieghem, A. Jamakovic, and R.E. Kooij. Effective graph resistance. *Linear Algebra and its Applications*, 435(10):2491–2506, nov 2011.

[28] Christoph Bandt and Bernd Pompe. Permutation entropy: A natural complexity measure for time series. *Physical Review Letters*, 88(17), apr 2002.

[29] N. Atienza, L. M. Escudero, M. J. Jimenez, and M. Soriano-Trigueros. Persistent entropy: a scale-invariant topological statistic for analyzing cell arrangements. 2019.

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

[31] Audun Myers and Firas A. Khasawneh. Delay parameter selection in permutation entropy using topological data analysis. *arXiv preprint:1905.04329*, 2020.

[32] Andrew M Fraser and Harry L Swinney. Independent coordinates for strange attractors from mutual information. *Physical review A*, 33(2):1134, 1986.

[33] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[34] Michał Melosik and W Marszalek. On the 0/1 test for chaos in continuous systems. *Bulletin of the Polish Academy of Sciences Technical Sciences*, 64(3):521–528, 2016.

[35] Müller Riedl, A Müller, and N Wessel. Practical considerations of permutation entropy. *The European Physical Journal Special Topics*, 222(2):249–262, 2013.

[36] Zhenhu Liang, Yinghua Wang, Gaoxiang Ouyang, Logan J Voss, Jamie W Sleigh, and Xiaoli Li. Permutation auto-mutual information of electroencephalogram in anesthesia. *Journal of Neural Engineering*, 10(2):026004, feb 2013.

[37] Matthew B Kennel, Reggie Brown, and Henry DI Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical review A*, 45(6):3403, 1992.

[38] David S Broomhead and Gregory P King. Extracting qualitative dynamics from experimental data. *Physica D: Nonlinear Phenomena*, 20(2-3):217–236, 1986.

## Appendix

Table 1: Available flows and maps in dynamic systems library module.

| Dissipative Flows | Conservative Flows | Driven Dissipative Flows | Maps |
|---|---|---|---|
| Lorenz Att. | Simple Driven | Driven Pendulum | Logistic |
| Rossler Att. | Nose-Hoover Osc. | Driven Van der Pol Osc. | Henon |
| Chua Circuit | Labyrinth Chaos | Shaw Van der Pol Osc. | Sine |
| Coupled Lorenz-Rossler | Henon-Heiles Osc. | Forced Brusselator | Tent |
| Coupled Rossler-Rossler | | Ueda Osc. | Linear Congruent |
| Double Pendulum | | Duffing's Two-well Osc. | Ricker's Pop. |
| Diffusionless Lorenz Att. | | Duffing Van der Pol Osc. | Gauss |
| Complex Butterfly | | Rayleigh-Duffing Osc. | Cusp |
| Chen's Att. | | | Pincher's |
| Hadley Att. | | | Sine-circle |
| ACT Att. | | | Lozi |
| Rabinovich-Fabrikant Att. | | | Delayed Logistic |
| Rigid Body Feedback | | | Tinkerbell |
| Moore-Spiegel Osc. | | | Burgers |
| Thomas Att. | | | Holmes |
| Halvorsen's Att. | | | Kaplan-Yorke |
| Burke-Shaw Att. | | | |
| Rucklidge Att. | | | |
| WINDMI | | | |
| Simple Quadratic Flow | | | |
| Simple Cubic Flow | | | |
| Simple Piecewise Flow | | | |
| Double Scroll | | | |

Table 2: Available functions, noise models, and medical data in dynamical systems library module.

| Functions | Noise Models | Medical Data |
|---|---|---|
| Sine | Gaussian | Electrocardiogram |
| Incommensurate Sine | Uniform | Electroencephalogram |
| | Rayleigh | |

Table 3: Parameter selection methods available in parameter selection module for both the delay and dimension parameters.

| Algorithm | Reference(s) | Dimension or Delay |
|---|---|---|
| Mutual Information | [32, 30] | Delay |
| Autocorrelation | [33, 30] | Delay |
| Frequency Analysis | [34, 30, 31] | Delay |
| Multi-scale Permutation Entropy | [35, 30] | Delay |
| Permutation Auto-mutual Information | [36, 30] | Delay |
| SW1PerS | [1, 31] | Delay |
| False Nearest Neighbors | [37, 30] | Dimension |
| Multi-scale Permutation Entropy | [35, 30] | Dimension |
| Singular Spectrum Analysis | [38, 30] | Dimension |