topopy Documentation

Dan Maljovec

Sep 11, 2021

Contents:

1 Installation 1

2 Example Usage 3

3 API Documentation 5

4 License 17

5 Indices and tables 19

Python Module Index 21

Index 23

CHAPTER 1

Installation

Currently, you can use pip to install this package and all of its prerequisite libraries:

```
pip install topopy
```

Or to install from source, install all of the prerequiste libraries:

- scipy
- numpy
- scikit-learn
- networkx
- nglpy

And then clone and build the source repository:

```
git clone https://github.com/maljovec/topopy.git
cd topopy
make
python setup.py [develop|install]
```

CHAPTER 2

Example Usage

```
import nglpy as ngl
import numpy as np
import topopy

def hill(_x):
    _x = np.atleast_2d(_x)
    x = _x[:, 0]
    y = _x[:, 1]
    return np.exp(- ((x - .55)**2 + (y-.75)**2)/.125) + 0.01*(x+y)

X = np.random.rand(100,2)
Y = hill(X)
graph = ngl.EmptyRegionGraph(beta=1.0, relaxed=False, p=2.0)

msc = topopy.MorseSmaleComplex(graph=graph,
                               gradient='steepest',
                               normalization='feature')
msc.build(X, Y)
msc.get_partitions()
```

CHAPTER 3

API Documentation

TopoPy - Topological constructs for Python.

TopoPy is a Python package for constructing approximate topological constructs in arbitrary dimensions using a neighborhood graph structure for approximating local gradient.

class topopy.**TopologicalObject**(*graph=None*, *gradient='steepest'*, *normalization=None*, *aggregator=None*, *debug=False*)

A base class for housing common interactions between Morse and Morse-Smale complexes, and Contour and Merge Trees

Parameters

- **graph** (*nglpy.Graph*) – A graph object used for determining neighborhoods in gradient estimation
- **gradient** (*str*) – An optional string specifying the type of gradient estimator to use. Currently the only available option is 'steepest'.
- **normalization** (*str*) – An optional string specifying whether the inputs/output should be scaled before computing. Currently, two modes are supported 'zscore' and 'feature'. 'zscore' will ensure the data has a mean of zero and a standard deviation of 1 by subtracting the mean and dividing by the variance. 'feature' scales the data into the unit hypercube.
- **aggregator** (*str*) – An optional string that specifies what type of aggregation to do when duplicates are found in the domain space. Default value is None meaning the code will error if duplicates are identified.
- **debug** (*bool*) – An optional boolean flag for whether debugging output should be enabled.
- **short_circuit** (*bool*) – An optional boolean flag for whether the contour tree should be short circuited. Enabling this will speed up the processing by bypassing the fully augmented search and only focusing on partially augmented split and join trees

static aggregate_duplicates(*X*, *Y*, *aggregator='mean'*, *precision=16*)

A function that will attempt to collapse duplicates in domain space, X, by aggregating values over the range space, Y.

Parameters

- **X** (*np.ndarray*) – An m-by-n array of values specifying m n-dimensional samples
- **Y** (*np.array*) – A m vector of values specifying the output responses corresponding to the m samples specified by X
- **aggregator** (*str*) – An optional string or callable object that specifies what type of aggregation to do when duplicates are found in the domain space. Default value is mean meaning the code will calculate the mean range value over each of the unique, duplicated samples.
- **precision** (*int*) – An optional positive integer specifying how many digits numbers should be rounded to in order to determine if they are unique or not.

Returns A tuple where the first value is an m'-by-n array specifying the unique domain samples and the second value is an m' vector specifying the associated range values. m' <= m.

Return type tuple(np.ndarray, np.array)

build(*X*, *Y*, *w=None*)

Assigns data to this object and builds the requested topological structure

Uses an internal graph given in the constructor to build a topological object on the passed in data. Weights are currently ignored.

Parameters

- **X** (*np.ndarray*) – An m-by-n array of values specifying m n-dimensional samples
- **Y** (*np.array*) – An m vector of values specifying the output responses corresponding to the m samples specified by X
- **w** (*np.array*) – An optional m vector of values specifying the weights associated to each of the m samples used. Default of None means all points will be equally weighted

Returns

Return type None

check_duplicates()

Function to test whether duplicates exist in the input or output space.

First, if an aggregator function has been specified, the domain space duplicates will be consolidated using the function to generate a new range value for that shared point. Otherwise, it will raise a ValueError. The function will raise a warning if duplicates exist in the output space

Returns

Return type None

get_dimensionality()

Returns the dimensionality of the input space of the input data

Returns Integer specifying the dimensionality of the input samples.

Return type int

get_neighbors(*idx*)

Returns a list of neighbors for the specified index

Parameters **idx** (*int*) – An integer specifying the query point

Returns Integer list of neighbors indices

Return type list of int

get_normed_x(*rows=None*, *cols=None*)
Returns the normalized input data requested by the user.

Parameters

- **rows** (*list of int*) – A list of non-negative integers specifying the row indices to return
- **cols** (*list of int*) – A list of non-negative integers specifying the column indices to return

Returns A matrix of floating point values specifying the normalized data values used in internal computations filtered by the three input parameters.

Return type np.ndarray

get_sample_size()
Returns the number of samples in the input data

Returns Integer specifying the number of samples.

Return type int

get_weights(*indices=None*)
Returns the weights requested by the user

Parameters **indices** (*list of int*) – A list of non-negative integers specifying the row indices to return

Returns An array of floating point values specifying the weights associated to the input data rows filtered by the indices input parameter.

Return type np.array

get_x(*rows=None*, *cols=None*)
Returns the input data requested by the user

Parameters

- **rows** (*list of int*) – A list of non-negative integers specifying the row indices to return
- **cols** (*list of int*) – A list of non-negative integers specifying the column indices to return

Returns A matrix of floating point values specifying the input data values filtered by the two input parameters.

Return type np.ndarray

get_y(*indices=None*)
Returns the output data requested by the user

Parameters **indices** (*list of int*) – A list of non-negative integers specifying the row indices to return

Returns An array of floating point values specifying the output data values filtered by the indices input parameter.

Return type np.array

load_data_and_build(*filename*, *delimiter=', '*)
Convenience function for directly working with a data file.

This opens a file and reads the data into an array, sets the data as an nparray and list of dimnames

Parameters **filename** (*str*) – string representing the data file

Returns

Return type None

reset()
Empties all internal storage containers

Returns

Return type None

class topopy.**MorseComplex**(*graph=None*, *gradient='steepest'*, *normalization=None*, *simplification='difference'*, *aggregator=None*, *debug=False*)
A wrapper class for the C++ approximate Morse complex Object

Parameters

- **graph** (*nglpy.Graph*) – A graph object used for determining neighborhoods in gradient estimation
- **gradient** (*str*) – An optional string specifying the type of gradient estimator to use. Currently the only available option is 'steepest'.
- **normalization** (*str*) – An optional string specifying whether the inputs/output should be scaled before computing. Currently, two modes are supported 'zscore' and 'feature'. 'zscore' will ensure the data has a mean of zero and a standard deviation of 1 by subtracting the mean and dividing by the variance. 'feature' scales the data into the unit hypercube.
- **simplification** (*str*) – An optional string specifying how we will compute the simplification hierarchy. Currently, three modes are supported 'difference', 'probability' and 'count'. 'difference' will take the function value difference of the extrema and its closest function valued neighboring saddle (standard persistence simplification), 'probability' will augment this value by multiplying the probability of the extremum and its saddle, and count will order the simplification by the size (number of points) in each manifold such that smaller features will be absorbed into neighboring larger features first.
- **aggregator** (*str*) – An optional string that specifies what type of aggregation to do when duplicates are found in the domain space. Default value is None meaning the code will error if duplicates are identified.
- **debug** (*bool*) – An optional boolean flag for whether debugging output should be enabled.

build(*X*, *Y*, *w=None*)
Assigns data to this object and builds the Morse Complex

Uses an internal graph given in the constructor to build a Morse complex on the passed in data. Weights are currently ignored.

Parameters

- **X** (*np.ndarray*) – An m-by-n array of values specifying m n-dimensional samples
- **Y** (*np.array*) – An m vector of values specifying the output responses corresponding to the m samples specified by X
- **w** (*np.array*) – An optional m vector of values specifying the weights associated to each of the m samples used. Default of None means all points will be equally weighted

Returns

Return type None

get_classification(*idx*)
Given an index, this function will report whether that sample is a local maximum or a regular point.

Parameters **idx** (*int*) – A non-negative integer less than the sample size of the input data.

Returns A string specifying the classification type of the input sample: will be 'maximum' or 'regular.'

Return type str

get_current_labels()
Returns a list of tuples that specifies the extremum index labels associated to each input sample

Returns a list of non-negative integers specifying the extremum-flow indices associated to each input sample at the current level of persistence

Return type list of tuple(int, int)

get_label(*indices=None*)
Returns the label indices requested by the user

Parameters **indices** (*list of int*) – A list of non-negative integers specifying the row indices to return

Returns A list of integers specifying the extremum index of the specified rows.

Return type list of int

get_merge_sequence()

Returns a data structure holding the ordered merge sequence of extrema simplification

Returns **dict of int** – A dictionary of tuples where the key is the dying extrema and the tuple is the the persistence, parent index, and the saddle index associated to the dying index, in that order.

Return type tuple(float, int, int)

get_partitions(*persistence=None*)
Returns the partitioned data based on a specified persistence level

Parameters **persistence** (*float*) – A floating point value specifying the size of the smallest feature we want to track. Default = None means consider all features.

Returns **dict of int** – A dictionary lists where each key is a integer specifying the index of the extremum. Each entry will hold a list of indices specifying points that are associated to this extremum.

Return type list of int

get_persistence()
Retrieves the persistence simplfication level being used for this complex

Returns Floating point value specifying the current persistence setting

Return type float

get_sample_size(*key=None*)
Returns the number of samples in the input data

Parameters **key** (*int*) – An optional integer specifying a max id used for determining which partition size should be returned. If not specified then the size of the entire data set will be returned.

Returns An integer specifying the number of samples.

Return type int

reset()
Empties all internal storage containers

Returns

Return type None

save(*filename=None*)
Saves a constructed Morse Complex in json file

Parameters **filename** (*str*) – A filename for storing the hierarchical merging of features and the base level partitions of the data

Returns

Return type None

set_persistence(*p*)
Sets the persistence simplfication level to be used for representing this complex

Parameters **p** (*float*) – A floating point value specifying the internally held size of the smallest feature we want to track.

Returns

Return type None

to_json()
Writes the complete Morse complex merge hierarchy to a string

Returns A string storing the entire merge hierarchy of all maxima

Return type str

class topopy.**MorseSmaleComplex**(*graph=None*, *gradient='steepest'*, *normalization=None*, *simplification='difference'*, *aggregator=None*, *debug=False*)
A wrapper class for the C++ approximate Morse-Smale complex Object

Parameters

- **graph** (*nglpy.Graph*) – A graph object used for determining neighborhoods in gradient estimation
- **gradient** (*str*) – An optional string specifying the type of gradient estimator to use. Currently the only available option is 'steepest'.
- **normalization** (*str*) – An optional string specifying whether the inputs/output should be scaled before computing. Currently, two modes are supported 'zscore' and 'feature'. 'zscore' will ensure the data has a mean of zero and a standard deviation of 1 by subtracting the mean and dividing by the variance. 'feature' scales the data into the unit hypercube.
- **simplification** (*str*) – An optional string specifying how we will compute the simplification hierarchy. Currently, three modes are supported 'difference', 'probability' and 'count'. 'difference' will take the function value difference of the extrema and its closest function valued neighboring saddle (standard persistence simplification), 'probability' will augment this value by multiplying the probability of the extremum and its saddle, and

count will order the simplification by the size (number of points) in each manifold such that smaller features will be absorbed into neighboring larger features first.

- **aggregator** (*str*) – An optional string that specifies what type of aggregation to do when duplicates are found in the domain space. Default value is None meaning the code will error if duplicates are identified.
- **debug** (*bool*) – An optional boolean flag for whether debugging output should be enabled.

build(*X*, *Y*, *w=None*)

Assigns data to this object and builds the Morse-Smale Complex

Uses an internal graph given in the constructor to build a Morse-Smale complex on the passed in data. Weights are currently ignored.

Parameters

- **X** (*np.ndarray*) – An m-by-n array of values specifying m n-dimensional samples
- **Y** (*np.array*) – An m vector of values specifying the output responses corresponding to the m samples specified by X
- **w** (*np.array*) – An optional m vector of values specifying the weights associated to each of the m samples used. Default of None means all points will be equally weighted

Returns

Return type None

get_classification(*idx*)

Given an index, this function will report whether that sample is a local minimum, a local maximum, or a regular point.

Parameters **idx** (*int*) – A non-negative integer less than the sample size of the input data.

Returns A string specifying the classification type of the input sample: will be 'maximum,' 'minimum,' or 'regular.'

Return type str

get_current_labels()

Returns a list of tuples that specifies the min-max index labels associated to each input sample

Returns a list of non-negative integer tuples specifying the min-max index labels associated to each input sample at the current level of persistence

Return type list of tuple(int, int)

get_label(*indices=None*)

Returns the label pair indices requested by the user

Parameters **indices** (*list of int*) – A list of non-negative integers specifying the row indices to return

Returns A list of integer 2-tuples specifying the minimum and maximum index of the specified rows, respectively.

Return type list of tuple(int, int)

get_merge_sequence()

Returns a data structure holding the ordered merge sequence of extrema simplification

Returns **dict of int** – A dictionary of tuples where the key is the dying extrema and the tuple is the the persistence, parent index, and the saddle index associated to the dying index, in that order.

Return type tuple(float, int, int)

get_partitions(*persistence=None*)
Returns the partitioned data based on a specified persistence level

Parameters **persistence** (*float*) – A floating point value specifying the size of the smallest feature we want to track. Default = None means consider all features.

Returns **dict of tuple(int,int)** – A dictionary lists where each key is a min-max tuple specifying the index of the minimum and maximum, respectively. Each entry will hold a list of indices specifying points that are associated to this min-max pair.

Return type list of int

get_persistence()
Retrieves the persistence simplfication level being used for this complex

Returns Floating point value specifying the current persistence setting

Return type float

get_sample_size(*key=None*)
Returns the number of samples in the input data

Parameters **key** (*int*) – An optional integer specifying a max id used for determining which partition size should be returned. If not specified then the size of the entire data set will be returned.

Returns An integer specifying the number of samples.

Return type int

get_stable_manifolds(*persistence=None*)
Returns the partitioned data based on a specified persistence level

Parameters **persistence** (*float*) – A floating point value specifying the size of the smallest feature we want to track. Default = None means consider all features.

Returns **dict of int** – A dictionary lists where each key is a integer specifying the index of the maximum. Each entry will hold a list of indices specifying points that are associated to this maximum.

Return type list of int

get_unstable_manifolds(*persistence=None*)
Returns the partitioned data based on a specified persistence level

Parameters **persistence** (*float*) – A floating point value specifying the size of the smallest feature we want to track. Default = None means consider all features.

Returns **dict of int** – A dictionary lists where each key is a integer specifying the index of the minimum. Each entry will hold a list of indices specifying points that are associated to this minimum.

Return type list of int

reset()
Empties all internal storage containers

Returns

Return type None

save(*filename=None*)
Saves a constructed Morse-Smale Complex in json file

Parameters **filename** (*str*) – A filename for storing the hierarchical merging of features and the base level partitions of the data

Returns

Return type None

set_persistence(*p*)
Sets the persistence simplfication level to be used for representing this complex

Parameters **p** (*float*) – A floating point value specifying the internally held size of the smallest feature we want to track.

Returns

Return type None

to_json()
Writes the complete Morse-Smale complex merge hierarchy to a string

Returns A string storing the entire merge hierarchy of all minima and maxima

Return type str

class topopy.**MergeTree**(*graph=None*, *gradient='steepest'*, *normalization=None*, *aggregator=None*, *debug=False*)
A wrapper class for the C++ merge tree data structure.

Parameters

- **graph** (*nglpy.Graph*) – A graph object used for determining neighborhoods in gradient estimation
- **gradient** (*str*) – An optional string specifying the type of gradient estimator to use. Currently the only available option is 'steepest'.
- **normalization** (*str*) – An optional string specifying whether the inputs/output should be scaled before computing. Currently, two modes are supported 'zscore' and 'feature'. 'zscore' will ensure the data has a mean of zero and a standard deviation of 1 by subtracting the mean and dividing by the variance. 'feature' scales the data into the unit hypercube.
- **aggregator** (*str*) – An optional string that specifies what type of aggregation to do when duplicates are found in the domain space. Default value is None meaning the code will error if duplicates are identified.
- **debug** (*bool*) – An optional boolean flag for whether debugging output should be enabled.

build(*X*, *Y*, *w=None*)
Assigns data to this object and builds the Merge Tree.

Uses an internal graph given in the constructor to build a merge tree on the passed in data. Weights are currently ignored.

Parameters

- **X** (*np.ndarray*) – An m-by-n array of values specifying m n-dimensional samples
- **Y** (*np.array*) – An m vector of values specifying the output responses corresponding to the m samples specified by X

- **w** (*np.array*) – An optional m vector of values specifying the weights associated to each of the m samples used. Default of None means all points will be equally weighted

Returns

Return type None

class topopy.**ContourTree**(*graph=None*, *gradient='steepest'*, *normalization=None*, *aggregator=None*, *debug=False*, *short_circuit=True*)

A class for computing a contour tree from two merge trees

Parameters

- **graph** (*nglpy.Graph*) – A graph object used for determining neighborhoods in gradient estimation
- **gradient** (*str*) – An optional string specifying the type of gradient estimator to use. Currently the only available option is 'steepest'.
- **normalization** (*str*) – An optional string specifying whether the inputs/output should be scaled before computing. Currently, two modes are supported 'zscore' and 'feature'. 'zscore' will ensure the data has a mean of zero and a standard deviation of 1 by subtracting the mean and dividing by the variance. 'feature' scales the data into the unit hypercube.
- **aggregator** (*str*) – An optional string that specifies what type of aggregation to do when duplicates are found in the domain space. Default value is None meaning the code will error if duplicates are identified.
- **debug** (*bool*) – An optional boolean flag for whether debugging output should be enabled.
- **short_circuit** (*bool*) – An optional boolean flag for whether the contour tree should be short circuited. Enabling this will speed up the processing by bypassing the fully augmented search and only focusing on partially augmented split and join trees

build(*X*, *Y*, *w=None*)

Assigns data to this object and builds the Contour Tree

Uses an internal graph given in the constructor to build a contour tree on the passed in data. Weights are currently ignored.

Parameters

- **X** (*np.ndarray*) – An m-by-n array of values specifying m n-dimensional samples
- **Y** (*np.array*) – An m vector of values specifying the output responses corresponding to the m samples specified by X
- **w** (*np.array*) – An optional m vector of values specifying the weights associated to each of the m samples used. Default of None means all points will be equally weighted

Returns

Return type None

get_seeds(*threshold*)

Returns a list of seed points for isosurface extraction given a threshold value

Parameters **threshold** (*float*) – The isovalue for which we want to identify seed points for isosurface extraction

Returns A list of integers representing seed points in the data held by this object. There will be one seed point for each connected component of the isosurface defined by the given threshold value.

Return type list of int

reset()
Empties all internal storage containers

Returns

Return type None

CHAPTER 4

License

BSD 3-Clause License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

topopy, 5

Index

A

aggregate_duplicates() (*topopy.TopologicalObject static method*), 5

B

build() (*topopy.ContourTree method*), 14
build() (*topopy.MergeTree method*), 13
build() (*topopy.MorseComplex method*), 8
build() (*topopy.MorseSmaleComplex method*), 11
build() (*topopy.TopologicalObject method*), 6

C

check_duplicates() (*topopy.TopologicalObject method*), 6
ContourTree (*class in topopy*), 14

G

get_classification() (*topopy.MorseComplex method*), 9
get_classification() (*topopy.MorseSmaleComplex method*), 11
get_current_labels() (*topopy.MorseComplex method*), 9
get_current_labels() (*topopy.MorseSmaleComplex method*), 11
get_dimensionality() (*topopy.TopologicalObject method*), 6
get_label() (*topopy.MorseComplex method*), 9
get_label() (*topopy.MorseSmaleComplex method*), 11
get_merge_sequence() (*topopy.MorseComplex method*), 9
get_merge_sequence() (*topopy.MorseSmaleComplex method*), 11
get_neighbors() (*topopy.TopologicalObject method*), 6
get_normed_x() (*topopy.TopologicalObject method*), 7
get_partitions() (*topopy.MorseComplex method*), 9
get_partitions() (*topopy.MorseSmaleComplex method*), 12
get_persistence() (*topopy.MorseComplex method*), 9
get_persistence() (*topopy.MorseSmaleComplex method*), 12
get_sample_size() (*topopy.MorseComplex method*), 9
get_sample_size() (*topopy.MorseSmaleComplex method*), 12
get_sample_size() (*topopy.TopologicalObject method*), 7
get_seeds() (*topopy.ContourTree method*), 14
get_stable_manifolds() (*topopy.MorseSmaleComplex method*), 12
get_unstable_manifolds() (*topopy.MorseSmaleComplex method*), 12
get_weights() (*topopy.TopologicalObject method*), 7
get_x() (*topopy.TopologicalObject method*), 7
get_y() (*topopy.TopologicalObject method*), 7

L

load_data_and_build() (*topopy.TopologicalObject method*), 7

M

MergeTree (*class in topopy*), 13
MorseComplex (*class in topopy*), 8
MorseSmaleComplex (*class in topopy*), 10

R

reset() (*topopy.ContourTree method*), 14
reset() (*topopy.MorseComplex method*), 10
reset() (*topopy.MorseSmaleComplex method*), 12
reset() (*topopy.TopologicalObject method*), 8

S

save() (*topopy.MorseComplex method*), 10
save() (*topopy.MorseSmaleComplex method*), 13
set_persistence() (*topopy.MorseComplex method*), 10
set_persistence() (*topopy.MorseSmaleComplex method*), 13

T

to_json() (*topopy.MorseComplex method*), 10
to_json() (*topopy.MorseSmaleComplex method*), 13
TopologicalObject (*class in topopy*), 5
topopy (*module*), 5